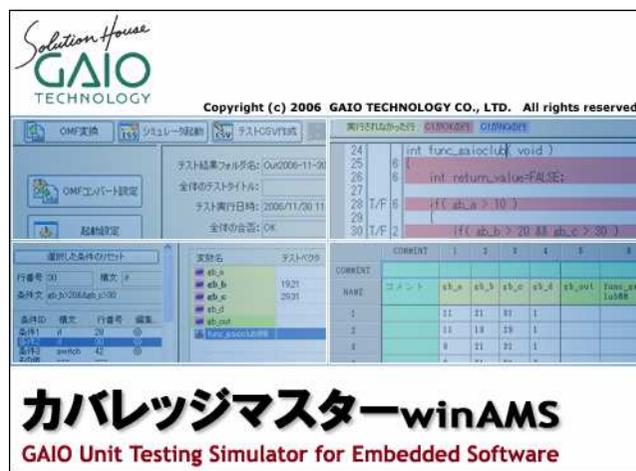




JaSST2007 C3-1 テクノロジーセッション

組み込み開発向けモジュール単体テストソリューション
CO/C1カバレッジテストデータ生成機能を搭載

カバレッジマスター winAMS



ガイオ・テクノロジー(株) 営業部



最初に: ガイオテクノロジーご紹介

25年来の組込みツールメーカー、自社に受託開発部隊もあり

- VAX/VMS対応の汎用クロス開発ツールでツールビジネスに参入
- 以後、EWS ~ PCのダウンサイジングを経て8000インストールベースの実績
- 近年はプロトタイピングツールやシミュレーション検証ツールなどを中心にビジネス展開

ガイオ・テクノロジー株式会社
GAIO TECHNOLOGY CO., LTD.

設立 1980年
資本金 2億9800万円
従業員 80名
本社: 横浜
事業所: 東京、松山、九州
子会社: GAIO INC(USA)



東京・日本橋



愛媛・松山



横浜・本社



九州:



組み込みのあらゆるツールを提供するメーカー

ガイオ・テクノロジーの開発ツール&ソリューション



**統合開発環境
クロスコンパイラ**

**システムシミュレータ
(MATLAB/Simulink連携 含む)**

**単体テストツール
(マイコンシミュレータ使用)**

実機自動テストシステム

＜ガイオの受託開発内容＞
ターゲットボードデバッグのためのエミュレーションシステム

本システムのメリット

- デバッグを前倒しすることによる、工数の削減
- 他部門の開発スピードに遅れに依存しない開発スタイルの確立
- 人海戦術的なデバッグ手法から、自動テスト方式へ

HMIプロトタイピング

SoCプロトタイピング

ドキュメント生成・ソース解析

チャート/該当ソース行を
ダイナミックリンク



単体テストを自動化する カバレッジマスターwinAMS

Solution House
GAIO
TECHNOLOGY

Copyright (c) 2006 GAIO TECHNOLOGY CO., LTD. All rights reserved.

実行結果ファイル名: OUt2006-11-20
全体のテストタイトル:
テスト実行日時: 2006/11/20 11
全体の合否: OK

```
24 int func_aaiocli( void )
25
26 int return_value=FALSE;
27
28 if( ab_a > 10 )
29
30 if( ab_b > 20 || ab_c > 30 )
```

行番号	積欠	行番号	編集
条件1	if	20	◎
条件2	if	21	◎
条件3	if	22	◎
条件4	if	23	◎
条件5	if	24	◎
条件6	if	25	◎
条件7	if	26	◎
条件8	if	27	◎
条件9	if	28	◎
条件10	if	29	◎
条件11	if	30	◎

行番号	積欠	行番号	編集
条件1	if	20	◎
条件2	if	21	◎
条件3	if	22	◎
条件4	if	23	◎
条件5	if	24	◎
条件6	if	25	◎
条件7	if	26	◎
条件8	if	27	◎
条件9	if	28	◎
条件10	if	29	◎
条件11	if	30	◎

カバレッジマスターwinAMS
GAIO Unit Testing Simulator for Embedded Software



ガイオ カバレッジマスター winAMS 特長

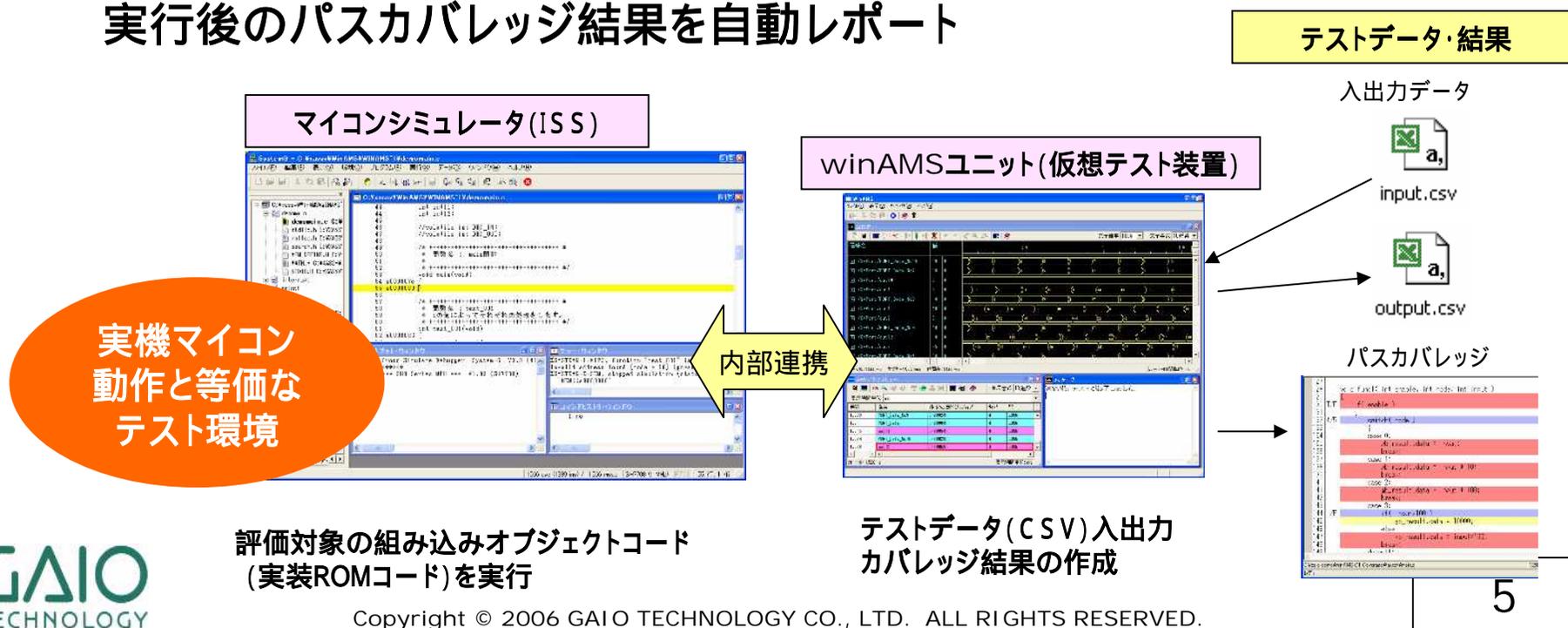
組み込みソフトの単体テストに特化した 評価ツール

マイコンシミュレータ(ISS)を使用して「実装ROMコード」をテスト

- 実際のマイコンの組み込みオブジェクトコードで単体テストを実行
- 対象の評価ソースコードの書き換えは一切不要

テストデータ(関数、変数名)は全てCSVファイルで入出力

実行後のパスカバレッジ結果を自動レポート





関数への入出力テスト方法(仕組み)

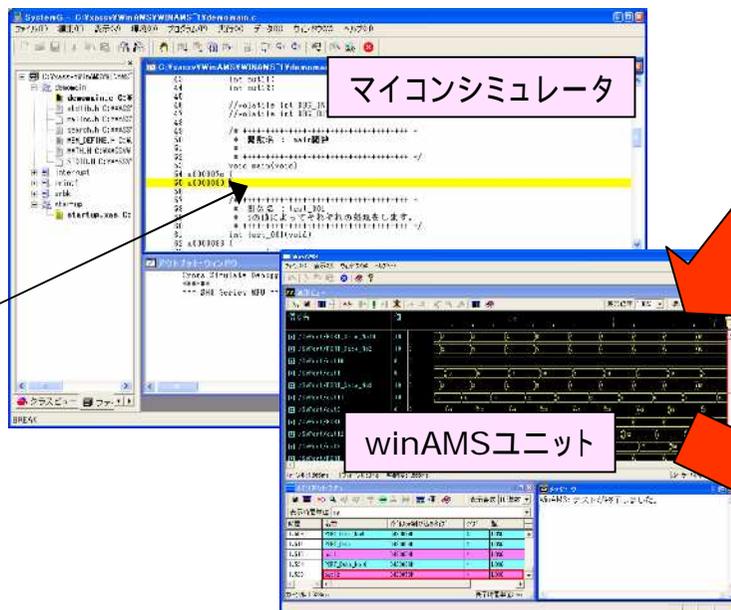
「カバレッジマスター winAMS」モジュール単体入出力テスト

- クロスコンパイルした 組み込みオブジェクトをそのまま使用
- 関数名、変数名、テストデータをCSVで入出力
- 期待値との照合結果を自動レポート

試験対象のソースコード

```
base(int a, int b, int c)
{
    if (a == 1)
    {
        if (b == 1)
        {
            idx = 0; // data[0]
            if (c == 1)
            {
                pos = 0; // data[0].str[0]
                :
            }
        }
    }
    // 結果の設定
    data[idx].c = data[idx].str[pos];
    value = data[idx].str[pos]; //- 1;
}
```

- ・組み込みソースをそのまま使用
- ・コンパイラも現在お使いのものを利用可能
- ・バッチ処理で自動テスト



カバレッジマスター-winAMS パッケージ

テスト入力データ CSV

	A	B	C	D
1	mod	base	base_test	3
2	base@a	base@b	base@c	value
3	1	1	1	2
4	1	2	2	15
5	2	1	1	26
6	2	1	2	27

- ・対象の関数名
- ・入力変数名&入力データ
- ・出力変数名&期待値

テスト結果出力 CSV

E	F	G	H	I	
5					
data[0].c	data[1].c	data[2].c	data[3].c		
2	0	0	0	OK	0.058ms
2	15	0	0	NG	0.062ms
2	15	26	0	NG	0.060ms
2	15	27	0	NG	0.062ms

- ・出力変数名&変数結果出力
- ・期待値との比較結果(OK or NG)

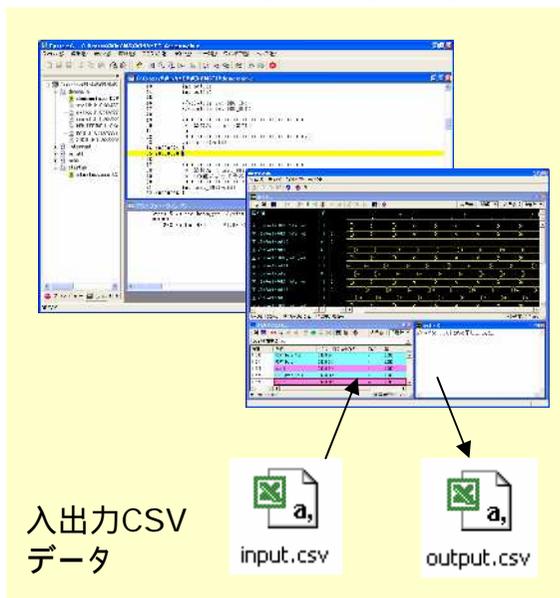


C0/C1カバレッジ自動レポート

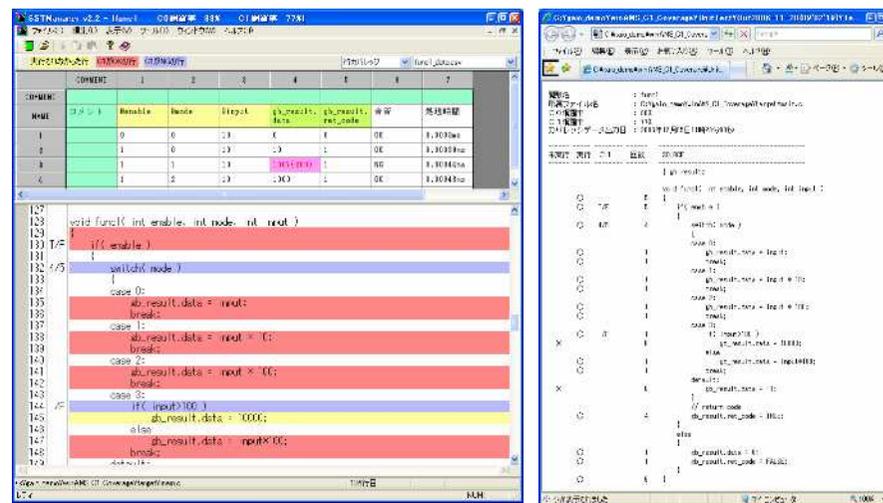
「カバレッジマスター winAMS」 C0/C1カバレッジ結果を自動レポート

- 条件分岐によるコードパスの網羅テストで品質を保証
- 「winAMS」により テスト実行後に 実行したソース行を色表示
- 組み込みソフトのC0/C1カバレッジの自動テストを実現

winAMS シミュレータ



C0/C1カバレッジ結果表示



クリックしたテストデータによる実行ソース行を色表示



呼び出し関数STUB作成機能

STUB関数作成・管理機能を装備

元のソースコード修正なしで 呼び出し関数の入れ替えが可能

- HWアクセス部分、システムコール、永久ループ関数など、テスト環境では実行できない部分を他の関数へ置き換え
- 関数リスト上で STUBのON/OFFを簡単に切り替え可能
- テスト対象の関数ソースコードは、一切修正不要

置換	ソースファイル名	関数名	スタブ関数名
<input type="checkbox"/>	irq.c	HANDLER_IRQ1	
<input type="checkbox"/>	irq.c	HANDLER_IRQ2	
<input type="checkbox"/>	main.c	func_c1_samp	
<input type="checkbox"/>	main.c	main	
<input type="checkbox"/>	main.c	func3_sub_calc	
<input type="checkbox"/>	main.c	func1	
<input type="checkbox"/>	main.c	func2	
<input type="checkbox"/>	main.c	func3	
<input type="checkbox"/>	main.c	func4	
<input checked="" type="checkbox"/>	main.c	func3_sub_read_io	AMSTB_func3_sub_read...

関数表示

スケルトン出力

スケルトン編集

セット

リセット



```

5
6 /* WINAMS_STUB[main.c:func3_sub_read_io]
7 int AMSTB_func3_sub_read_io( void )
8 {
9     return 1;|
10 }
11

```

STUB関数のスケルトンコードを自動作成
この中に テスト時の仮の関数を記述

クロスコンパイラでコード化
実際のアプリケーションコードとリンクして実行

オブジェクト内の関数リスト
ここで置き換えるSTUB関数を作成・切り替え



ポインタ変数対応・その他機能

ポインタ変数・引数にも対応

- 変数の実体を伴わないポインタ変数・引数の場合でもそのままテスト可能
- 入力データCSVの変数名に「\$」を付けるだけで 実体を自動割り当て
- 割り付けエリアは MPUのメモリモデルから 自由に指定可能



```
int func1( char *data )
```

	A	B	C	D	E
1	mod	func1	タイトル	2	1
2	\$func1@data	func1@data[0]	func1@@		
3		1 0x1f		0	
4		1 0x20		1	
5		1 0x21		0	

ポインタ変数の実体を作るための割り当てエリアを指定

ポインタ割り当てエリア設定

エリアを設定する

0x60000 ~

0x60030

CSVファイルに「~」で 期待値の範囲指定も可能

2	value	data[0].c
3	100~200	2
4	100~200	2



CSVテストデータ形式(参考)

入力テストデータは資産化が可能

- マイコンに依存しない入力データフォーマット
- モジュールのテスト資産として再利用が容易
- 一般的なCSV形式でファイル保存が可能

mod	func	コメント	4	2	
input	TAB.mode	array[2]	func@a	func@@	output
1	1	-1	-1	1	2
12	43	56	78	1~8	0x4
0x10	0x23	0x66	0x11		

評価する関数名

入力変数名
(関数名@ は引数)

出力変数名
(@@関数名は戻り値)

入力データ

期待値データ
(空でも可)

テスト関数サンプル

```
// 入力データ
char input;
struct {
    char mode;
    int flag;
} TAB;
char array[3] = { '1','2','3' };

// 結果データ
int output;

// 評価対象の関数 func
int func(int a)
{
    if (a == 1) {
        if (b == 1) {
            | 中略
        }
        return 1;
    }
}
```

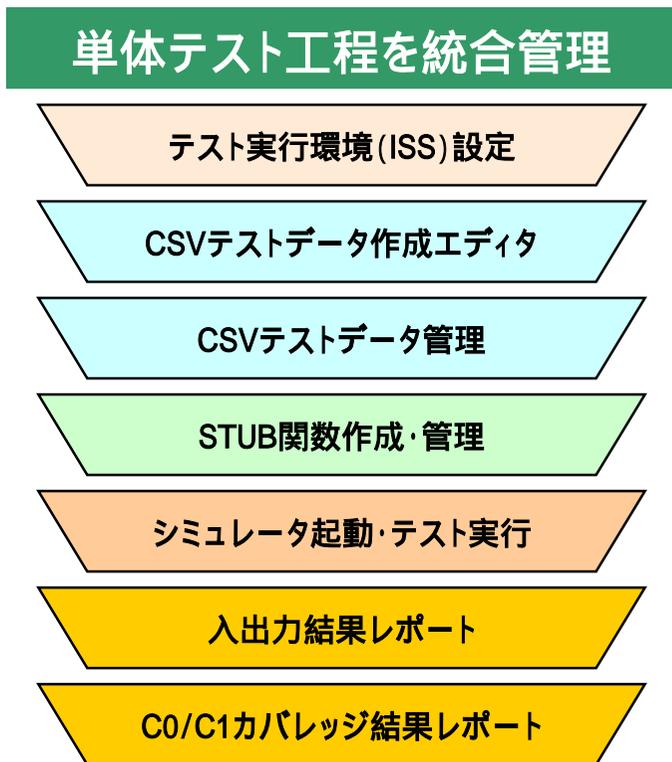


テストデータ作成支援 CO/C1入力データ自動作成機能



テストデータ管理・実行ツール SSTManager

テストデータ作成から テスト実行までを統合管理するツール



V2.2にて
インターフェースを
一新しました！



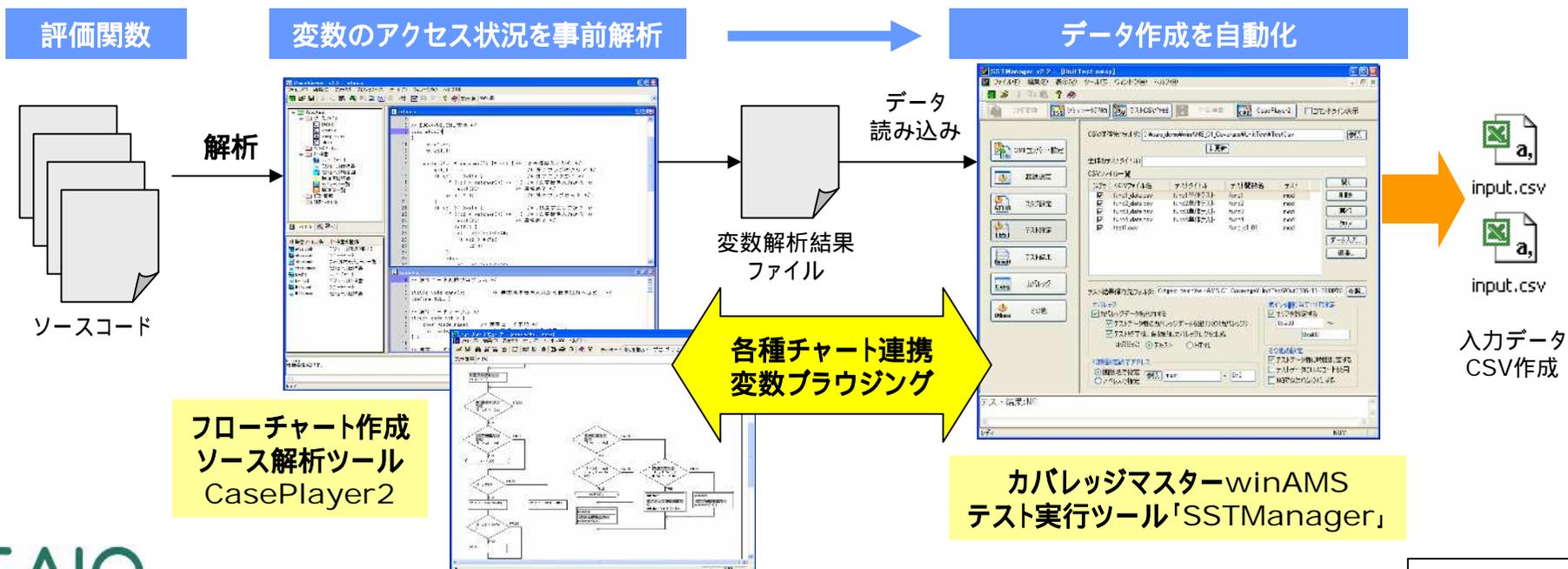
CasePlayer2と連携したデータ作成支援

ソース解析ツール「CasePlayer2」と連携して テストデータ作成を支援

- テスト対象の関数が使用する変数を事前に解析

コード解析によりC1カバレッジ入力データ作成を自動化

- C1カバレッジを満たす最小限のテスト入力データを自動生成
- ユーザーは、生成された入力データに対する結果を、関数仕様を元に確認するだけ





CasePlayer2解析結果を利用(1)

評価対象の関数が参照/代入している外部変数を自動検索

- 入力条件となる変数(引数、外部変数)を自動検索し コード解析工数を省力化

関数が参照している外部変数
(単体テストの入力変数)

関数が書き換えている外部変数
(単体テストの評価変数)

関数の引数リスト
(単体テストの入力変数)

The screenshot shows the '変数一覧' (Variable List) window with several tabs: '変数一覧', '入力・出力変数', 'レジスタマップ', and '定義名リスト'. The '入力・出力変数' tab is active, displaying a tree view of variables categorized into '入力変数' (Input Variables), '出力変数' (Output Variables), '関数内スタティック' (Function Internal Static), '引数' (Arguments), and 'リターン値' (Return Values). Under '入力変数', variables 'gb_a', 'gb_b', and 'gb_c' are listed. Under '出力変数', 'gb_out' is listed. Under '引数', 'func_c1_01@code' and 'func_c1_01@@" are listed. To the right, there are two panels: 'INPUT' and 'OUTPUT'. The 'INPUT' panel contains a list with '@code', 'gb_a', 'gb_b', and 'gb_c', with 'gb_c' selected. The 'OUTPUT' panel contains a list with 'gb_out' and 'func_c1_01@@" with 'gb_out' selected. Below these panels are buttons for '削除' (Delete), '変更' (Change), 'I/O作成' (I/O Creation), and 'クリア' (Clear). At the bottom of the main window, there are '表示更新' (Update Display) and '表示オプション' (Display Options) sections. The '表示オプション' section has three checkboxes: 'サブ関数の入出力変数も表示する' (unchecked), '関数の入出力変数にスタブの設定を反映する' (unchecked), and '使用していないメンバは表示しない' (checked).

入出力変数リスト
評価対象の関数を指定すると
自動表示される

リストアップされた変数から
評価に必要な変数を
ユーザーが選択・指定



CasePlayer2解析結果を利用(2)

C1カバレッジを満たす入力データを自動生成

- 関数内の条件文をリストアップして、各関数のC1全ケースを網羅するデータを生成

```
// C1説明用サンプル
int gb_a, gb_b, gb_c, gb_d, gb_out;
char gb_unused1, gb_unused12;

int func_c1_01( int code )
{
    int return_value=FALSE;

    if( gb_a > 10 )
    {
        if( gb_b > 20 && gb_c > 30 )
        {
            gb_out = 0;
        }
        else
        {
            gb_out = -1;
        }
        return_value = FALSE;
    }
    else
    {
        switch( code )
        {
            case 1:
                gb_out = 1;
                break;
            case 2:
                gb_out = 2;
                break;
            case 3:
                gb_out = 3;
                break;
            default:
                gb_out = -1;
                break;
        }
        return_value = FALSE;
    }
    return return_value;
}
```

gb_b と gb_c の境界値 20、30の±1の値をデータとして生成

変数名	テストベクタ	I/O
@code		入力
gb_a		入力
gb_b	19,21	入力
gb_c	29,31	入力
gb_out		出力
func_c1_01@@		出力

条件文に関わる変数を自動検索 & ハイライト表示

switch文の変数codeの全ケースを生成

変数名	テストベクタ	I/O
@code	3,2,1,4	入力
gb_a		入力
gb_b		入力
gb_c		入力
gb_out		出力
func_c1_01@@		出力



最小限の組み合わせデータを生成

条件文のネスト状態を解析して C1を満たす重複の少ないデータを生成

- 各条件文 (if, switch など) に対して生成したデータの組み合わせを自動生成
- C0の観点から、1度生成した分岐経路を通るデータは生成しない 最適化を実施
- 必要最小限のデータ組み合わせを生成
- 前頁のソースサンプルでは わずか6個のデータ () でC0/C1を満たすことが可能

前頁サンプルで自動生成した入力データ

	COMMENT	1	2	3	4	5	6
COMMENT							
NAME	コメント	@code	εb_a	εb_b	εb_c	εb_out	func_c1_01 @@
1		1	11	21	31		
2		1	11	19	29		
3		1	9	21	31		
4		2	9	21	31		
5		3	9	21	31		
6		4	9	21	31		

ここは
期待値欄

EXCELの横桁データ数制限(256まで)を排除した
CSVデータエディタを搭載



選択したテストデータを一齐自動実行

管理ツールSSTManagerで選択したデータを自動実行

- 指定したCSVデータ(関数)を自動バッチ実行
- 結果表示まで 全て自動実行

全体のテストタイトル: オブジェクト内の全関数の一齐自動テスト

CSVファイル一覧

実行	CSVファイル名	テストタイトル	テスト関数名	テスト...
<input checked="" type="checkbox"/>	func1_data.csv	func1単体テスト	func1	mod
<input checked="" type="checkbox"/>	func2_data.csv	func2単体テスト	func2	mod
<input checked="" type="checkbox"/>	func3_data.csv	func3単体テスト	func3	mod
<input checked="" type="checkbox"/>	func4_data.csv	func4単体テスト	func4	mod
<input checked="" type="checkbox"/>	test1.csv		func_c1_01	mod
<input type="checkbox"/>	TestData.csv		func_c1_01	mod

テストデータを選択

テスト結果保存先フォルダ: C:\gaio_demo\winAMS_C1_Coverage\UnitTest\Out2006-11-28(09'02'19)

ISS シミュレータ起動

テスト開始ボタン

自動テスト実行

入出力テスト結果表示

テスト結果フォルダ名: Out2006-11-28(09'02'19)

全体のテストタイトル: オブジェクト内の全関数の一齐自動テスト

テスト実行日時: 2006/12/06 10:37:12

全体の合否: NG

テスト結果情報

テストデータファイル名	テストタイトル	テスト関...	判定
func1_data.csv	func1単体テスト	func1	NG
func2_data.csv	func2単体テスト	func2	OK
func3_data.csv	func3単体テスト	func3	NG
func4_data.csv	func4単体テスト	func4	No Check
test1.csv		func_c1_01	No Check

CO/C1カバレッジ結果表示

テスト結果フォルダ名: Out2006-11-28(09'02'19)

全体のテストタイトル: オブジェクト内の全関数の一齐自動テスト

テスト実行日時: 2006/12/06 10:47:26

全体のCOカバレッジ率: 97%

全体のC1カバレッジ率: 95%

テスト関数	COカバレッジ率	C1カバレッジ率	関数一覧	COカバレッジ率	C1カバレッジ率
func1	88%	77%	func3_sub_calc	100%	100%
func2	100%	100%			
func3	100%	100%			
func4	100%	100%			
func_c1_01	100%	100%			

ダブルクリックで各結果詳細表示へ



C0/C1カバレッジ、入出力結果表示

C0/C1カバレッジビュー

- カバレッジ結果とテストデータを表示
- 期待値と異なるデータセルを表示
 - 100?(200)
期待値が200だが結果は100
- C0を満たさない未実行行を表示
- C1を満たさない条件を明示
- 各行を通過するデータを解析し
 で表示
 - 未実行行のデータを追加する際の解析に非常に役立つ機能

COMMENT	1	2	3	4	5	6	7	
NAME	コメント	#enable	#mode	#input	gb_result.data	gb_result.ret_code	合否	処理時間
1		0	0	10	0	0	OK	0.0003ms
2		1	0	10	10	1	OK	0.00039ms
3		1	1	10	100?(200)	1	NG	0.00044ms
4		1	2	10	1000	1	OK	0.00046ms
5		1	3	10	1000	1	OK	0.00052ms


```

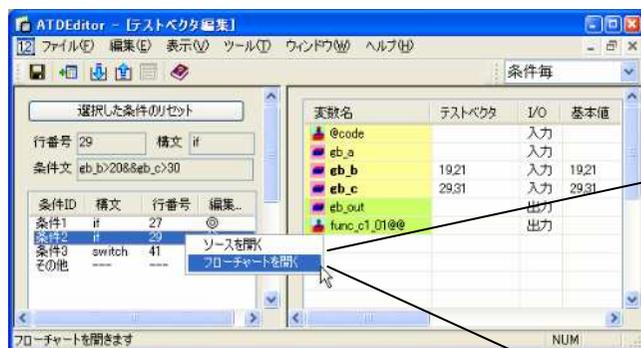
128 void func1( int enable, int mode, int input )
129 {
130     if( enable )
131     {
132         switch( mode )
133         {
134             case 0:
135                 gb_result.data = input;
136                 break;
137             case 1:
138                 gb_result.data = input * 10;
139                 break;
140             case 2:
141                 gb_result.data = input * 100;
142                 break;
143             case 3:
144                 if( input > 100 )
145                     gb_result.data = 10000;
146                 else
147                     gb_result.data = input * 100;
148                 break;
149             default:
150                 gb_result.data = -1;
151         }
152         // return code
153         gb_result.ret_code = TRUE;
154     }
  
```



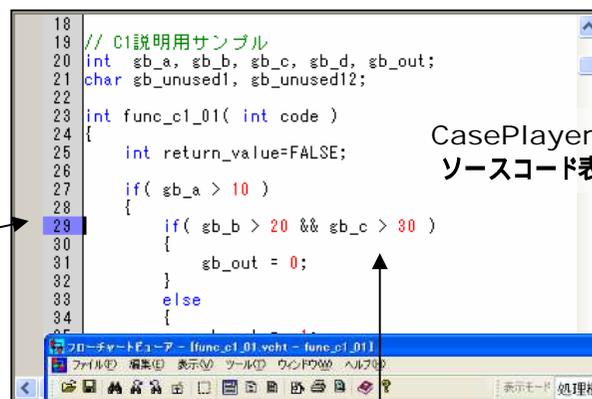
ソース解析に役立つチャートリンク機能

CasePlayer2が生成した各種チャート、変数参照リストへリンク

- 面倒な設定やソースコードの修正追加なしで簡単にチャートを生成
- 入出力データ作成時のロジック解析、パス解析を効率化

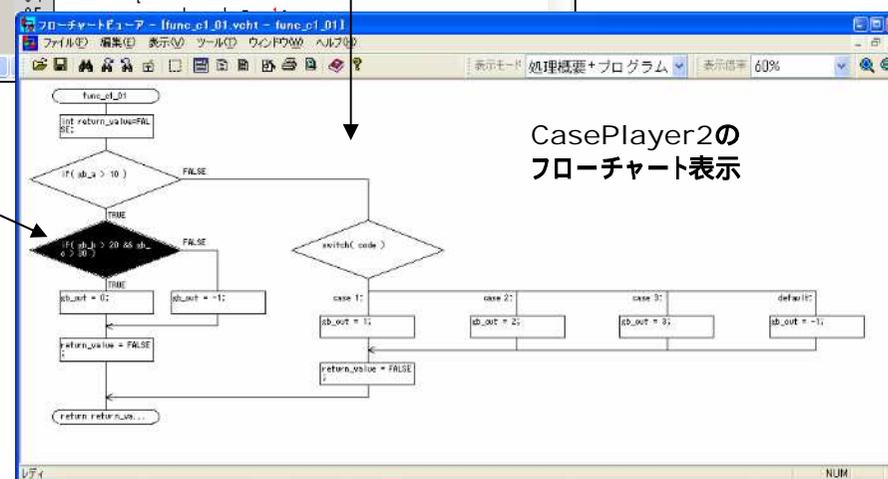


カバレッジマスターの変数設定画面



CasePlayer2の
ソースコード表示

リンク



CasePlayer2の
フローチャート表示



カバレッジマスターの優位性と導入効果



ICEによる手作業が どのように改善されるか

ICEを使用して、1関数60分で単体テストを行っている場合を例にとると・・・
作業時間の内訳は・・・

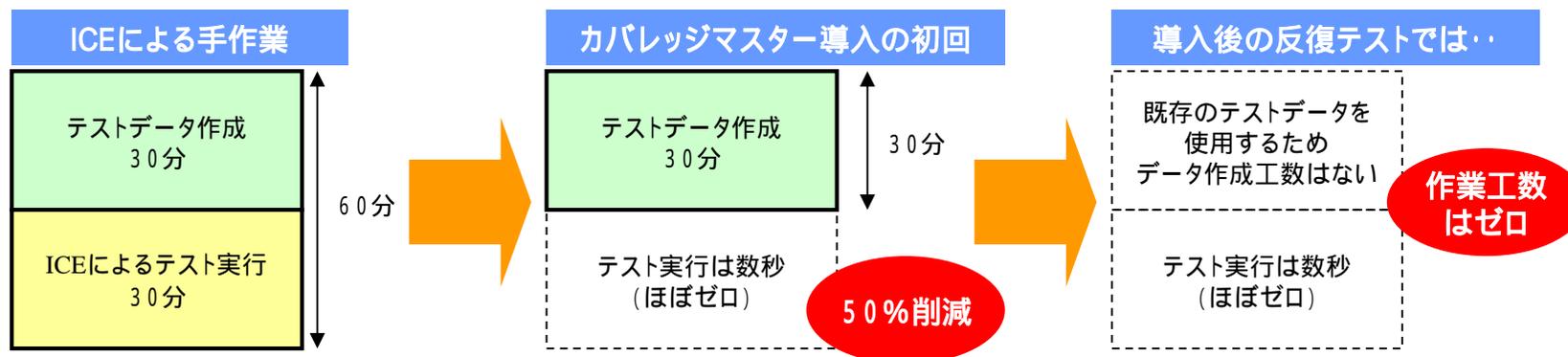
- ソース解析&テストデータ作成: 30分
- ICEによる手作業でのテスト実行: 30分

カバレッジマスターの導入により、初回のテストにおいては・・・

- ソース解析&テストデータ作成: 30分 同じ時間がかかる
- カバレッジマスターで自動実行: 数秒 テストは自動化され作業工数はゼロ

2回目以降の反復テストでは

- ソース解析&テストデータ作成、テスト実行の作業工数もゼロになる

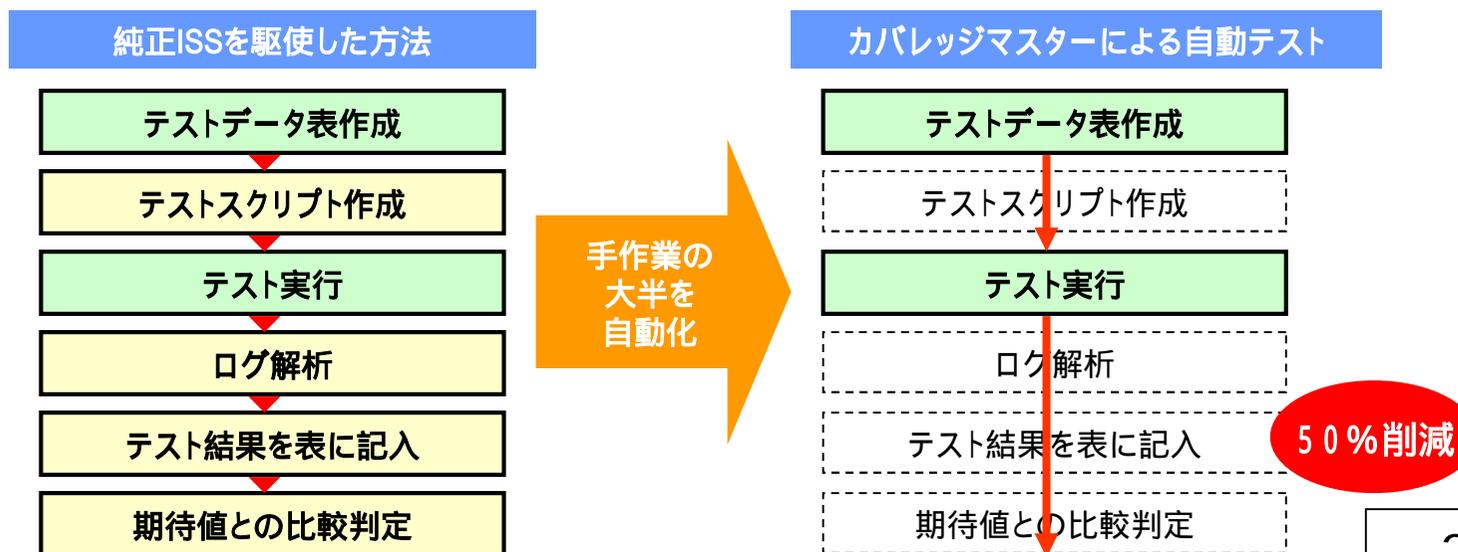




他社ISSを使用した場合との比較

他社(半導体メーカー純正)ISSを使用した単体テストとは

- ISS(デバッガ)がサポートするスクリプト言語を駆使して単体テストを行う方法
 - 関数の呼び出し指定、変数に対するデータ設定を行うスクリプトを作成
 - テスト実行後に実行ログファイルを解析して、評価対象の変数の値を確認
 - テスト結果を表に記入(手作業)
 - 期待値との比較判定(手作業)
- カバレッジをとることは困難、または膨大な手作業が必要
 - ステップ実行毎に、プリントしたソースコードをマーカーで塗りつぶす手作業となる





単体テストツールの必要条件

組み込みモジュール単体テストのテストツール選定には、以下の事項を満たす必要があります

項目	必要条件
工数	モジュール単体テストに必要な工数を低く抑えられるか
カバレッジ	網羅率を数値で正しく計測できるか
精度/信頼性	実機コード(コンパイラのバグも含む)を対象としたテスト実施が可能か
ソース書き換え	単体テスト用にソース修正作業が発生しないか
データ資産化	同じモジュールをテストする場合、テストデータの再利用が可能か
データ入力支援	膨大な工数が発生するテストデータを作成する際の支援機能があるか





カバレッジマスターのアドバンテージ

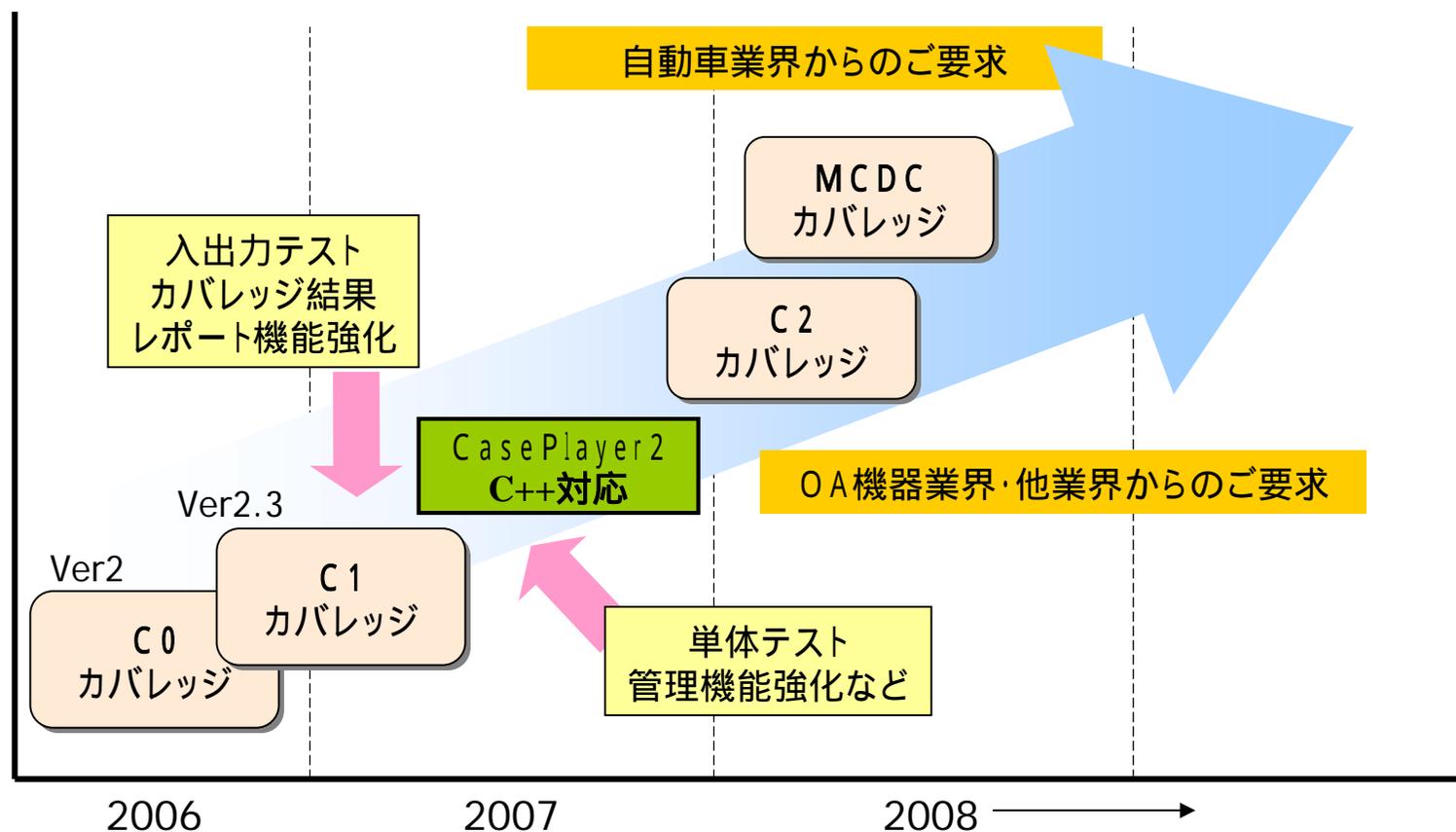
従来の単体テストツールの問題点をクリア

比較項目	工数	カバレッジ	精度 信頼性	ソース 書き換え	データの 資産化	テスト データ入 力支援	全体的 所見
	変数データ 設定などの 手作業	カバレッジ トレース	実機コードで のテスト	単体テスト用 に、ソース コードを修正	再帰テストの 自動化	テストデータ 作成の支援 機能	
ICE + デバッガ	× 全て手作業	× 事実上困難	実機コード	不要	× 毎回手作業	×	<ul style="list-style-type: none"> 精度は高いが人海戦術的要素が多く、多大な工数が発生する テストデータの資産化ができないため、再帰テストの際には工数がそのままn倍必要になる カバレッジ計測には不向き
純正ISS + スクリプト			実機コード	不要	× メーカー毎に 変更必要	×	<ul style="list-style-type: none"> 精度は高いが単体テスト用のスクリプト(仕掛け)を作成する必要がある MPU毎の汎用性がない カバレッジ計測には不向き
Windows系 (オープン系) ツール			× Cのロジック レベルテスト	× 必要			<ul style="list-style-type: none"> 汎用性は高いがオブジェクトが86系で実行されるため、ロジックのみのテストになり、最終ROMコードでのテストにはならない。 NATIVEコンパイラでコンパイルするためのソース修正工数が発生する。
カバレッジマ スター-winAMS	解析機能を利用 して効率化	自動レポート	実機コード	不要	全マイコン共 通のCSV形式	C0/C1テスト データ自動生 成	従来の単体テストツールの 問題点をクリア



カバレッジマスター-winAMS ロードマップ

カバレッジ機能、テストデータ生成、テスト管理など





テスト代行サービスについて



単体テスト代行サービスの概要

ガイオの優位性を活かして単体テストを実施

- 単体テストツールメーカーとしてのツール知識の活用
- コンパイラメーカーとしてのプログラム知見の活用

お客様に代わってテスト業務を完全代行

- テスト作業のアウトソーシング
- 第三者評価機関としての立場で品質評価

コンパイラメーカー
(プログラム知見)

テストツールメーカー
(ツール知見)

テスト代行におけるガイオの優位性

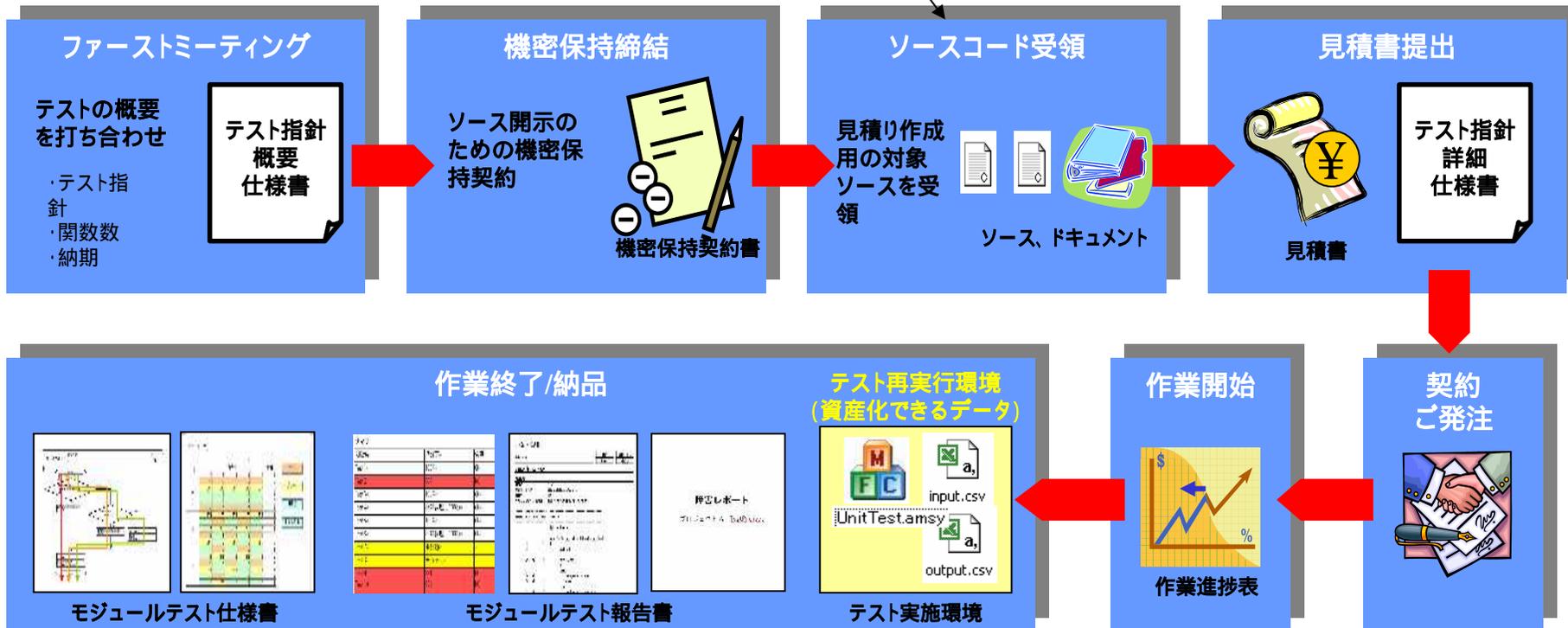




単体テスト代行の流れ

最終納品までの流れと 各フェーズでの受け渡し書類

- お見積もりはご提供頂くソースコード、ドキュメントにより決定





納品物: テスト仕様書とテストデータ

テスト指針とテストデータの意図を明確化

- カバレッジ指針 (C0、C1、C2など) に従って、分岐に対するテストデータを作成
- フローチャート上で、どの分岐パスを通過するデータかを明確化
- 納品後のお客様のレビューを容易にするためのドキュメントを作成

フローチャート

テストデータ毎の実行パスを表示

入力データ

		最小値	最大値	閾値 (+-1)		
パス		a	b	c	d	e
	1	1	1	0	1	10
	1	1	1	1	0	10
	1	1	1	1	1	10
	1	1	1	2	1	10
①	1	1	255	1	1	10
	1	1	1	0	0	10
	1	1	1	0	1	10
	1	1	1	1	1	10
	1	1	1	2	10	1
	1	1	1	255	10	1
	1	1	1	0	0	10
	1	1	1	0	0	10
	1	1	1	0	2	10
	1	1	1	0	255	10
	1	1	1	0	10	10

カバレッジ指針に従った
テストデータの意味を色で識別

どの実行パスを通過するデータが表示



納品物: テスト結果報告書

単体テストの実行結果についてのドキュメント

単体テスト結果

入力データ					期待値	合否
a	b	c	d	e		
1	1	0	0	10	1	OK
1	1	1	0	10	1	OK
1	1	1	0	10	1	OK
1	1	2	0	10	1	OK
1	1	255	1	10	1	OK
1	1	1	0	10	1	OK
1	1	0	1	10	1	OK
1	1	1	1	10	1	OK
1	1	1	2	10	1	OK
1	1	1	255	10	1	OK
1	1	0	0	10	1	OK
1	1	2	0	10	1	OK
1	1	255	0	10	1	OK
1	1	1	0	10	1	OK
1	1	1	2	10	1	OK
1	1	1	255	10	1	OK

カバレッジデータ

関数名 : func4
 所属ファイル名 : C:\winAMS_CM1\target\main.c
 網羅率 : 100%
 カバレッジデータ出力日 : 2006年06月07日17時18分27秒

```

-----
未実行  実行   回数   SOURCE
-----
int func4( int arg_flag )
    15  {
        static int default_val = -1;

    15  if( arg_flag == 1 )
        {
    7      if( gb_input < 10 )
            {
    2          gb_output = gb_input+10;
            }
    5      else if( gb_input < 20 )
            {
    3          gb_output = gb_input;
            }
        else
            {
    2          gb_output = default_val;
    2          return FALSE;
            }
    5      }
    8  else if( arg_flag == 2 )
        {
    7      if( gb_valA )
    1          gb_output = gb_input;
        else
    6          gb_output = -gb_input;
    7      }

    12  return TRUE;
    15  }
  
```



納品物: 障害報告書

単体テストで障害が発生した関数のドキュメント

障害発生関数サマリ

障害発生関数リスト		
関数名	網羅率	合否
Test2	88%	NG
Test6	75%	NG

各関数毎の単体テスト結果を一覧表でレポート

障害発生テスト結果報告書

テスト結果			
入力データ		期待値	合否
@a	@b	out2	
7	1	30?(10)	NG
7	5	20	OK
7	0x7fffffff	30	OK
7	0x80000000	30	OK
0x7fffffff	0	40	OK

網羅率 : 88%

各関数毎の入出力結果 (OK or NG) 期待値との照合結果レポート



END

最新の製品情報は

<http://www.gaio.co.jp/>



ガイオ・テクノロジー株式会社

会社名・商品名は各社の商標または登録商標です。
本資料の無断転載、複写はお断りします。

ガイオ・テクノロジー株式会社

日本橋事業所 営業部

〒103 東京都中央区日本橋人形町3-12-8

TEL. (03) 3662-3041

FAX. (03) 3662-3043

Email info@gaio.co.jp ..ご質問はこちらをお願いします。