シミュレーションを活用した
組み込みソフトウエアテストの改善

2012年1月26日

アルプス電気㈱ AUTO事業本部
第4技術部ファーム設計グループ
千葉　修一

美しい電子部品を究めます

ALPS®

Contents

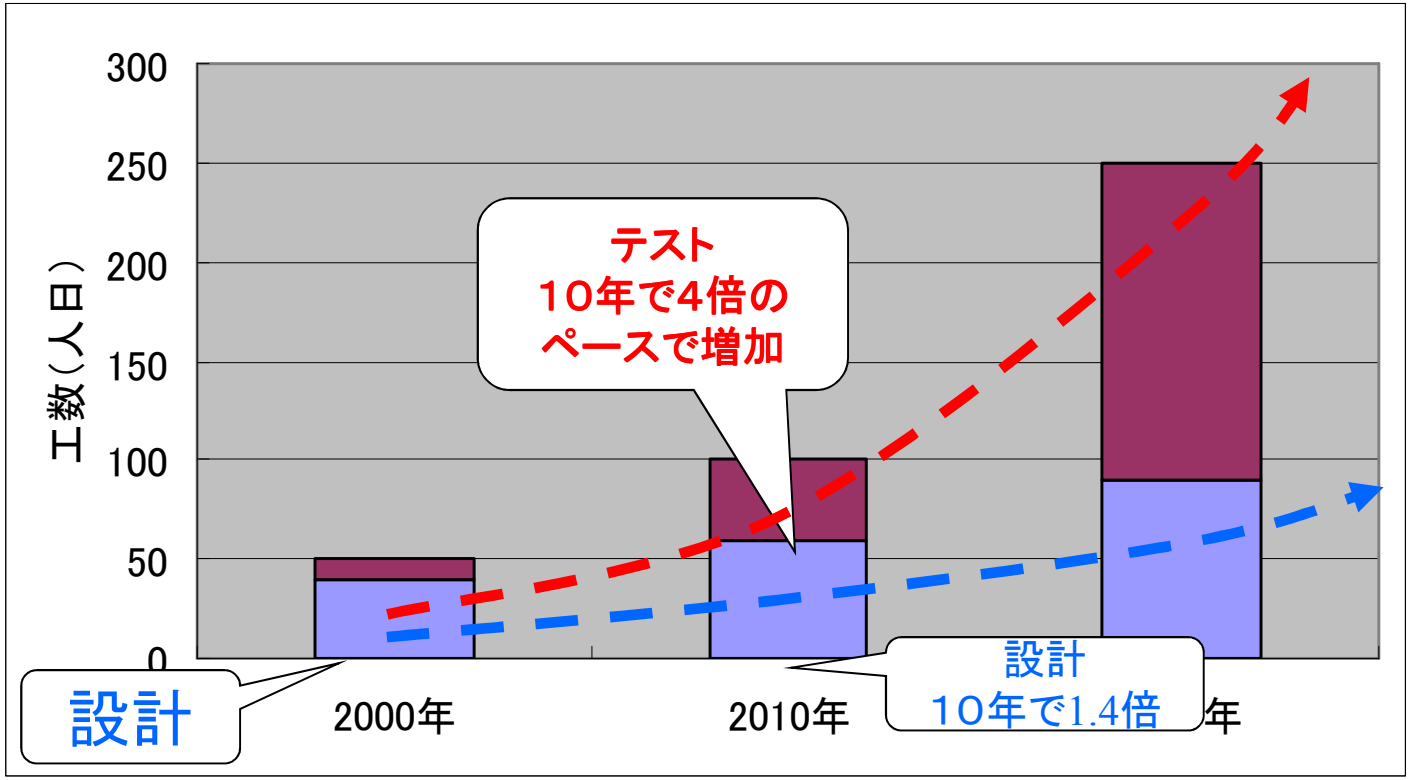
Contents

会社紹介

ECU組込み製品

Contents

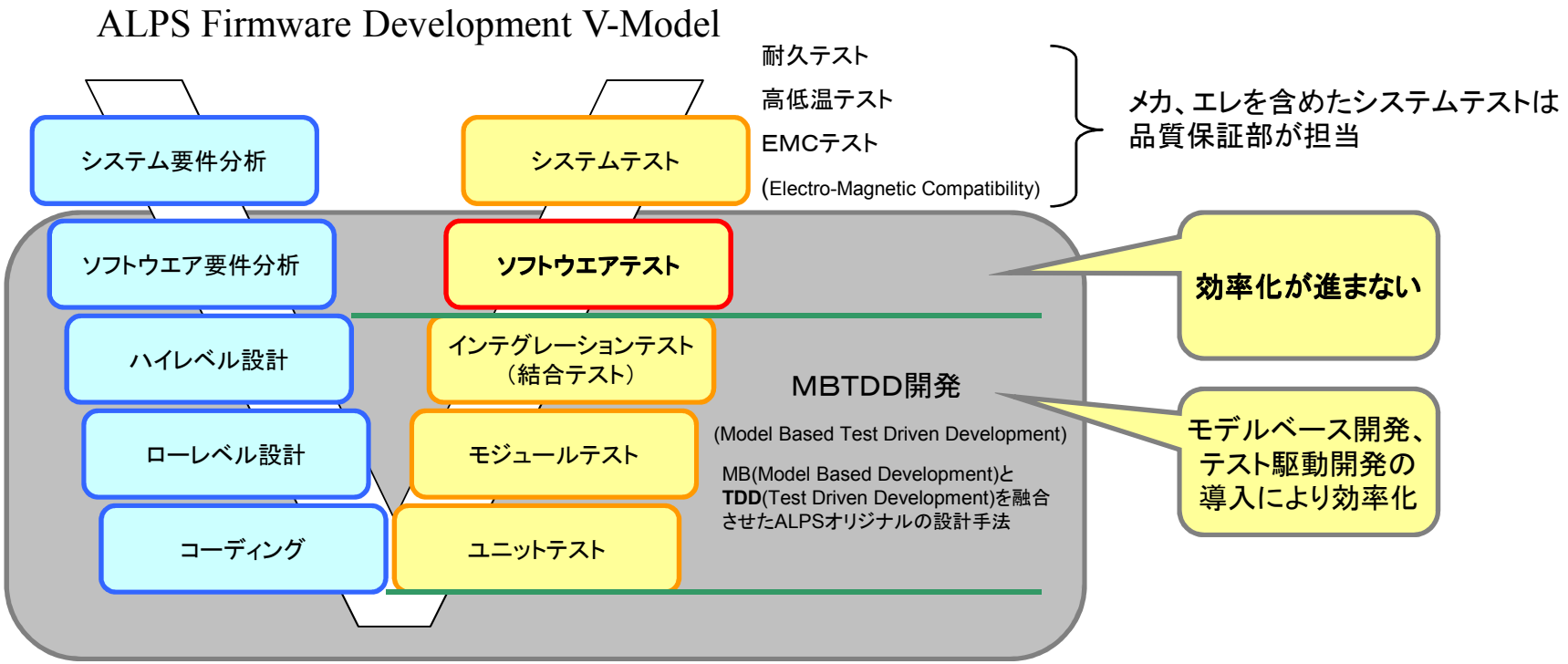
ソフトウエアテスト工数の爆発



背景

- ◆ ソフトウエアの規模自体が爆発的に増加。（ハイブリット、EV化）
- ◆ 設計は効率化できているが、テストは効率化できてない。

ソフトウエアテストの位置づけ

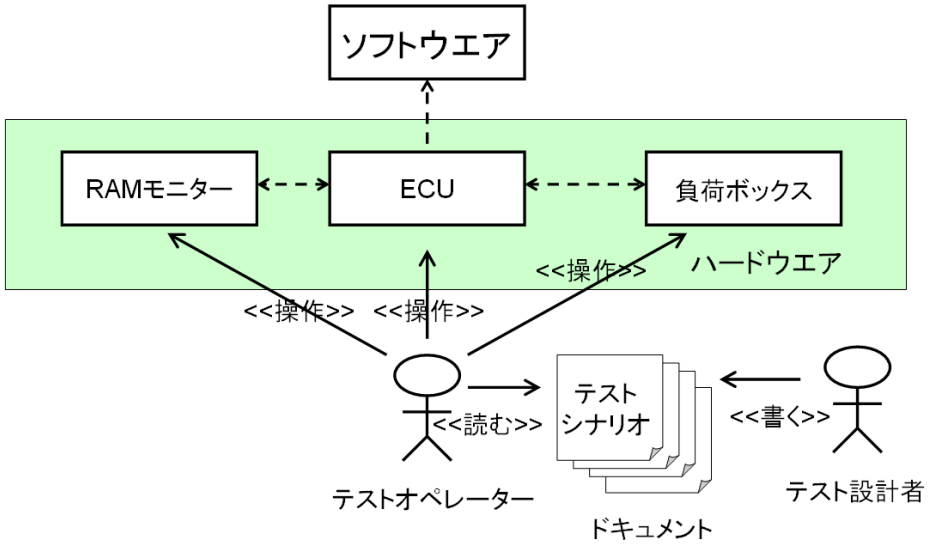


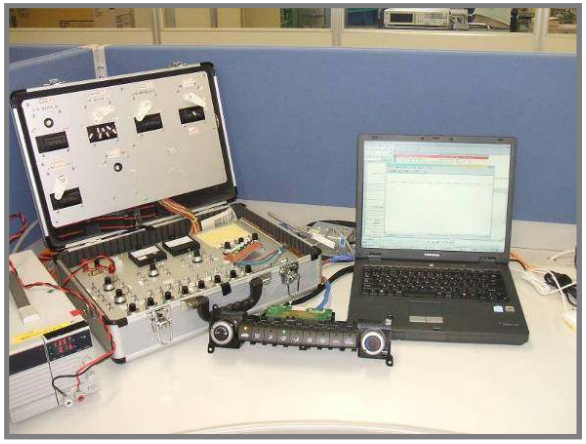
◆ テスト実施工数の爆発
車は量産までに何度も試作を繰り返すが、その度に膨大なテストが発生。

◆ テスト設計工数の爆発
仕様がどんどん複雑になりテスト設計が年々難しくなってきた。

テスト実施効率化の壁

従来テストの環境

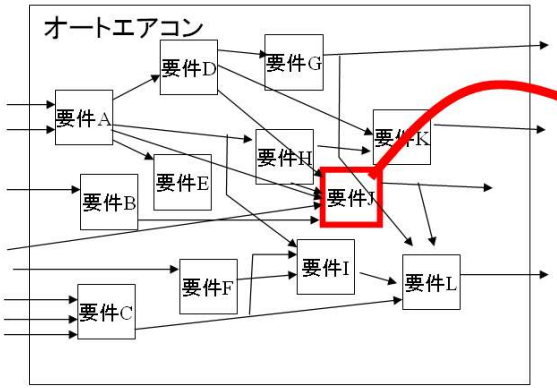


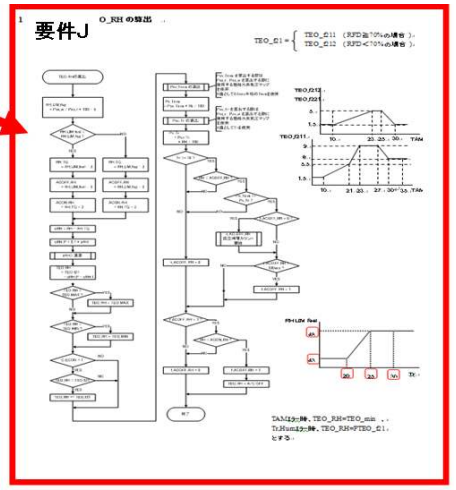


◆ 膨大なテスト数を人手をかけて行うので工数が増える一方。

◆ 自動化したいがインターフェースがハードウエアなので自動化に大きなコストがかかる。

テスト設計効率化の壁





◆ システムが複雑になり顧客要件の多くがシステムの内部変数を入出力として定義されている。
また、要件自体も非常に複雑。

◆ このような要件のテスト設計を効率良く行う手法がない。

Contents

自動化技術の検討結果

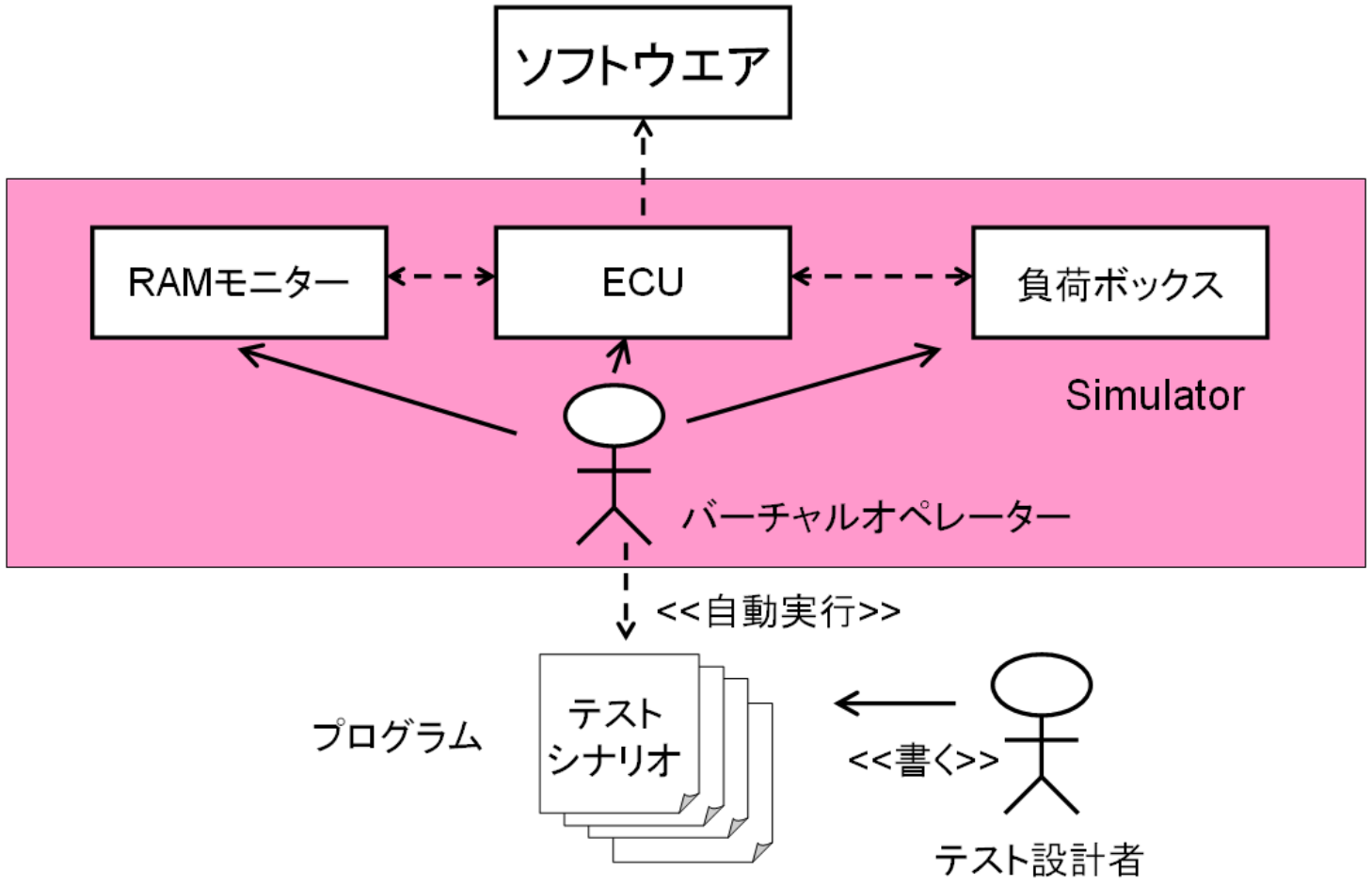
自動化の技術	コスト	品質	難易度	総合判定	備考
ロボット	△	△	○	△	携帯電話のテスト等で実績あり。 HMI入力は可能だが、ECU負荷ボックスのアナログ入力をボリュームを回して変化させるような微妙な操作は出来ない。
HILS	△	△	○	△	エンジンコントロールECU等で実績あり。 負荷ボックスの変わりにはなるが、ボタンを押すようなHMI操作は出来ない。
ロボット＋HILS	×	○	△	△	HMIと負荷ボックスの両方の操作の自動化が可能
SILS	○	○	△	○	ハードウエアの評価は出来ないが、ソフトウエアに限れば細かいテストまで自動化出来る。

HILS：Hardware In the Loop Simulation
SILS：Software In the Loop Simulation

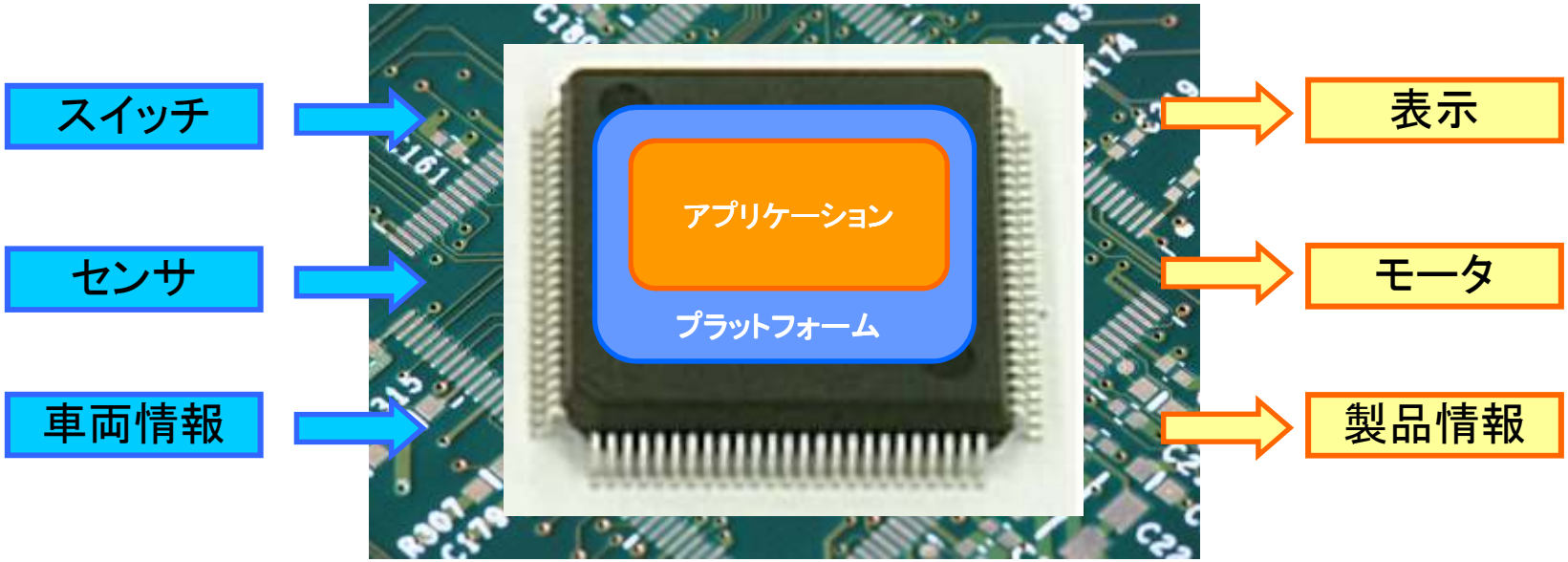
SILSを採用することに決定。
SILSを使ったテストでは長いので
SBT(Simulation Based Test)と呼ぶ事にした。

SBT(Simulation Based Test)概要

シミュレーションテストの環境



ソフトウエアアーキテクチャーの見直し(1/2)

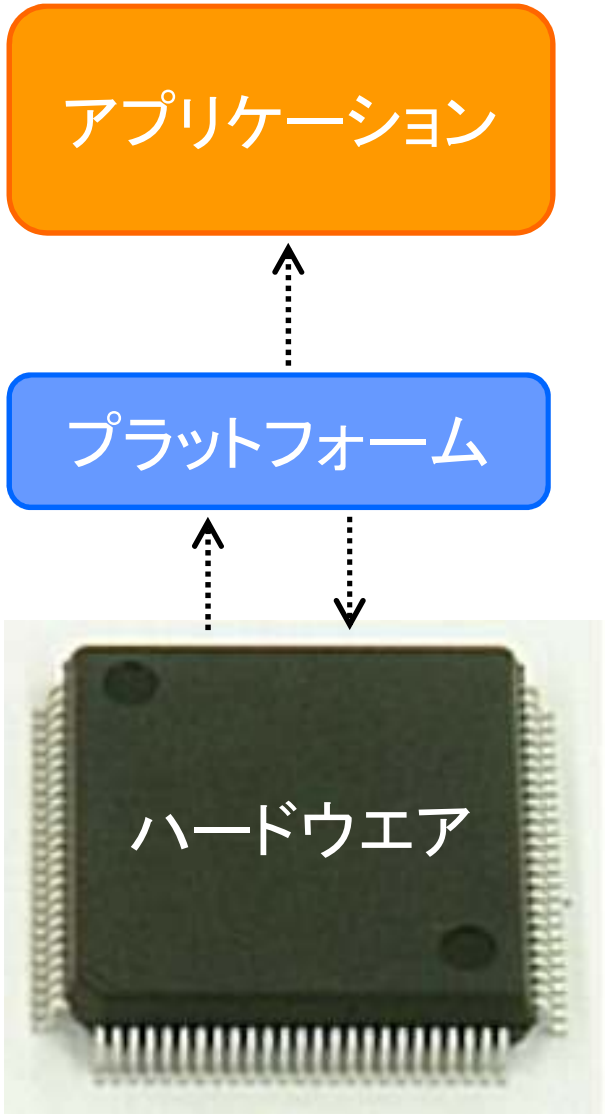


■ アプリケーション:客先要件の実装

■ プラットフォーム:ハードウエア要件の実装

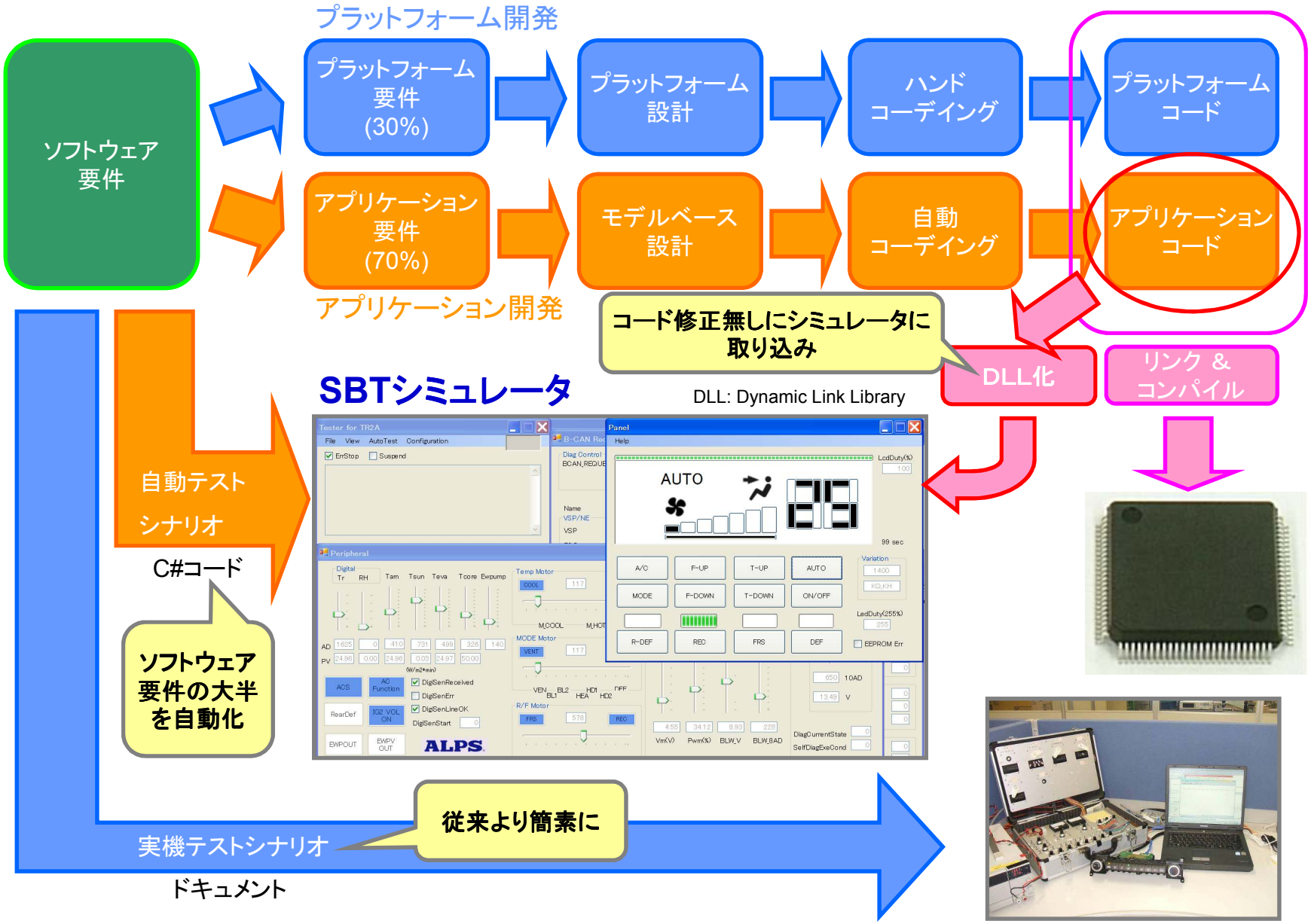
ソフトウエアをアプリケーションとプラットフォームに明確に分離

ソフトウエアアーキテクチャーの見直し(2/2)

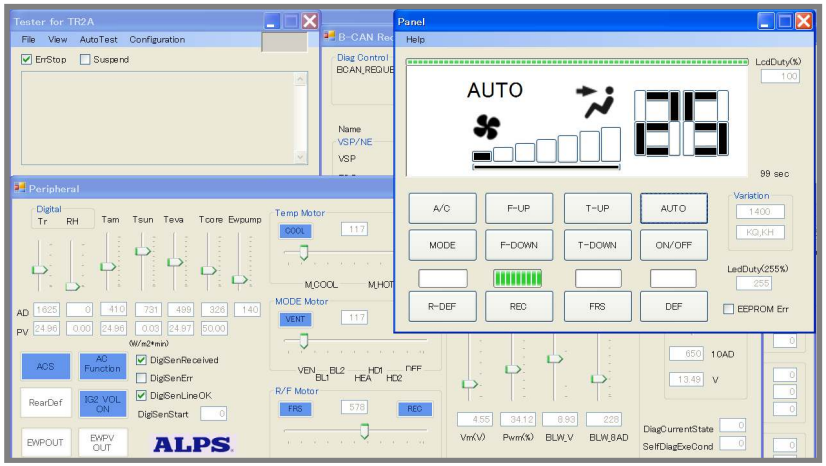


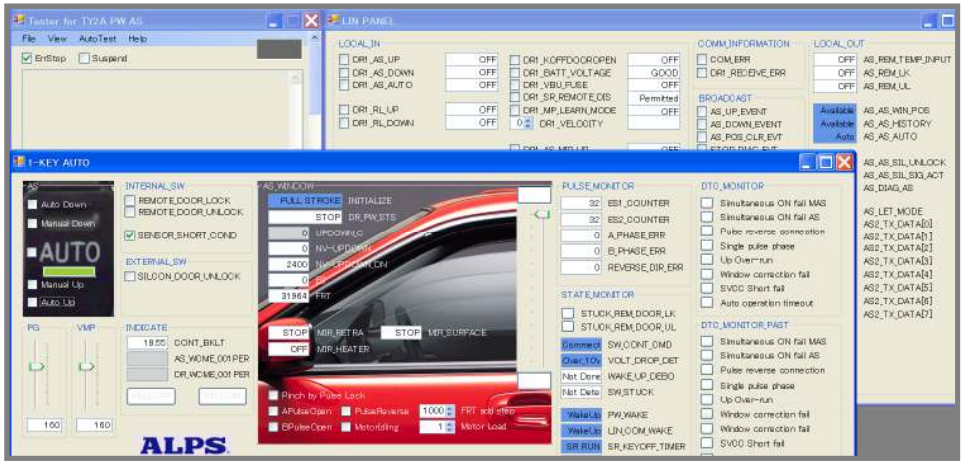
アプリケーションをハードウエアや処理系に「非依存」とすることで、一切手を加えることなくPC上でシミュレーションできるようした。

SBTのプロセス


プラットフォーム開発
ソフトウェア要件
プラットフォーム要件 (30%)
プラットフォーム設計
ハンドコーディング
プラットフォームコード
アプリケーション要件 (70%)
モデルベース設計
自動コーディング
アプリケーションコード
アプリケーション開発
コード修正無しにシミュレータに取り込み
DLL化
リンク & コンパイル
DLL: Dynamic Link Library
SBTシミュレータ
自動テストシナリオ
C#コード
ソフトウェア要件の大半を自動化
実機テストシナリオ
ドキュメント
従来より簡素に

SBTの特徴





■ MBTDD開発とリンクしたテストツール
テストツール用へのプログラム変換が不要、テスト環境の構築が容易。

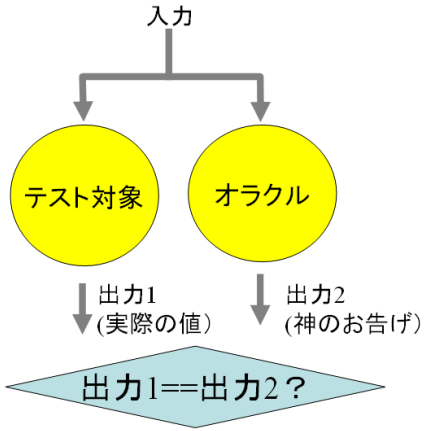
■ アプリケーションCコードを一切変更しないで使用することで
「SBTによるテスト = アプリケーションのテスト」を実現 。

■ これまでのテスト環境をそのままシミュレータに実装することで
違和感なく、テスト環境の移行が可能。

■ フレキシブルな対応
シミュレータを自分たちで開発することでフレキシブル、且つ継続した
改善が可能。

Contents

テストオラクル概要

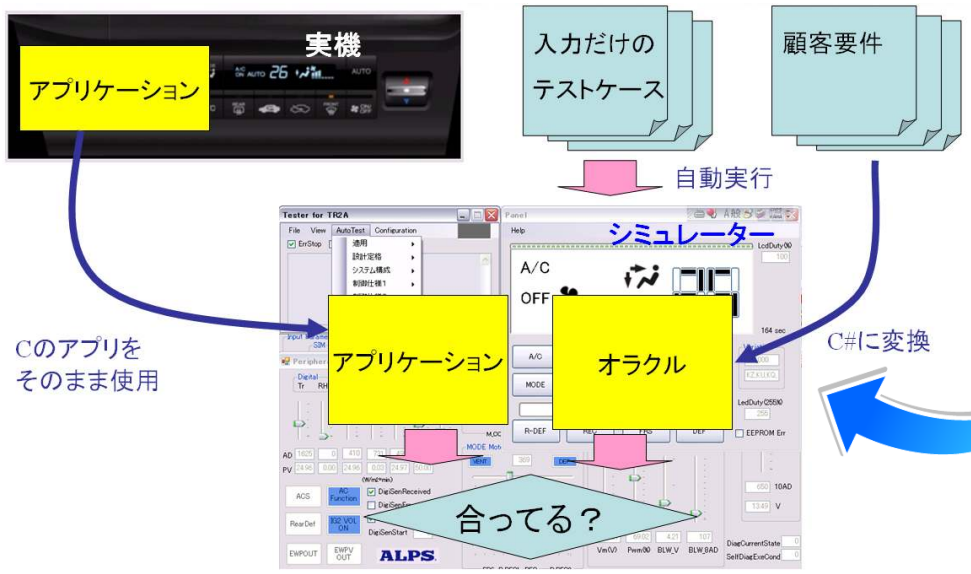


Oracleとは神のお告げという意味。

ファームウエアテストでは入力に対する出力の関係が正しいかどうかの判断基準になる仕組み。
簡単に言えば動く要件。

データベースソフト会社のオラクルとは関係ない。

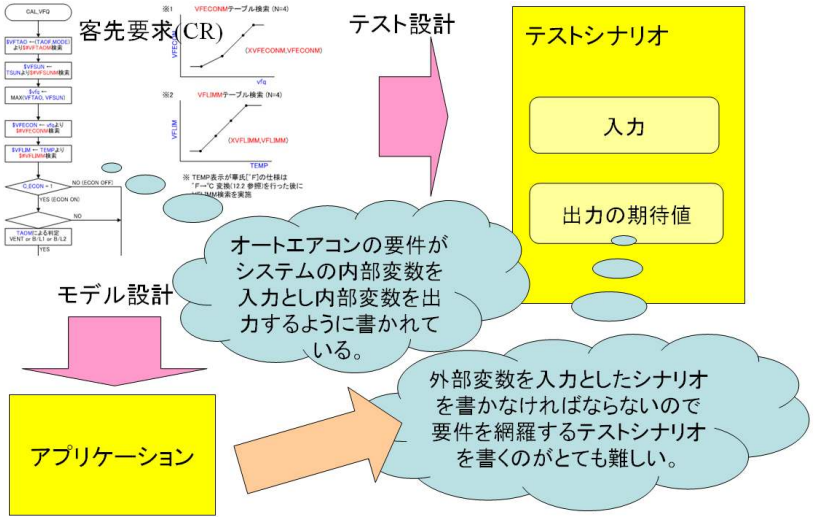
強力な言語C#で楽に実装。



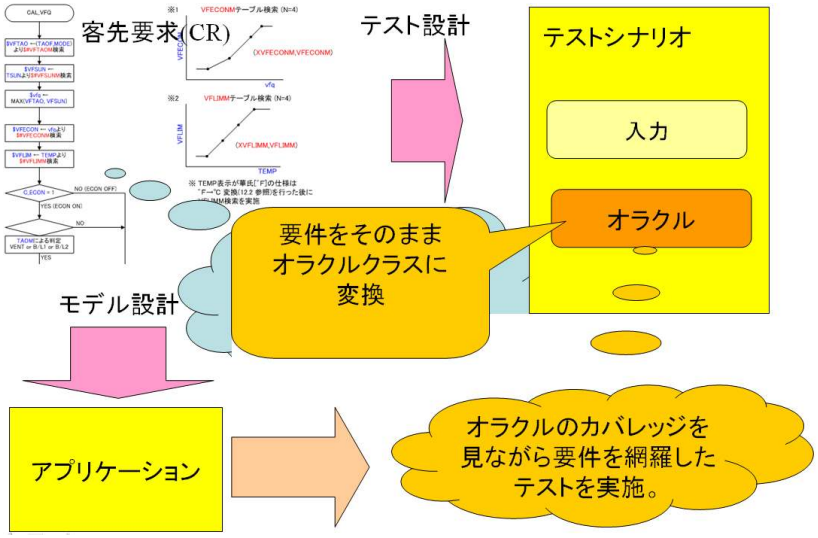

```
// ------------------------------------------------------------------------
//  TeoClrModel class
// ------------------------------------------------------------------------
public class TeoClrModel
{

        // TEO_MAXの決定(D12-007)
        if (fOnlyCmp)
        {
            TEOMAX = Param.EE_S16_AC_TEO_MAX_ECMP/100.0;
        }
        else
        {
            TEOMAX = Param.EE_S16_AC_TEO_MAX_BCMP/100.0;
        }

        // TEOCLRNMLの算出
        if (_TaoDirection == true)
        {
            if (Tao < _PreTao)
            {
                _TaoDirection = false;
            }
            else
            {
                TeoClrNml = CalcTeoClrNml(Tao, _TaoDirection);
            }
        }
        else
        {
            if (Tao > _PreTao)
            {
                _TaoDirection = true;
            }
            else
            {
```

テストオラクルの導入

オラクル非適用



オラクル適用



テストケース(入力作成)

```
// ------------------------------------------------------------------------------
// Case 0: ECOM = OFF時のチェック
// Condition: ECON = OFF
// Operation: Temp Change
// CheckPoint: VFQ = MAX(VF-TAO, VF-SUN)
// ------------------------------------------------------------------------------
for (int ccnt = 0; ccnt < 3; ccnt++)
{
    ushort vari;
    switch (ccnt)
    {
        default:
        case 0: vari = T_VARIATION_KZ_KU_KO_KH; break;
        case 1: vari = T_VARIATION_KC_KX;       break;
        case 2: vari = T_VARIATION_KA_KL;       break;
    }
    ResetSystem(vari);

    Can.EcoSet(false);

    double err = 0.1;
    for (int cnt = 0; cnt < 60; cnt++)
    {
        if (cnt < 20)
            { TempUpOnOff(); }
        else
            { TempDownOnOff(); }

        TestEquals(SelectVfQ(vari), (App.S16_FN_VF_Q_10MYget()/100.0), err);
        TestEquals(VfQModel._vfecon, (App.S16_FN_VF_ECON_10MYget()/100.0), err);
        TestEquals(VfQModel._vflim, (App.S16_FN_VF_LIM_10MYget()/100.0), err);
    }
}
TestPassed("Case 0");
```

	A	C	D	G	H	I	J
1	VF-Q test						
2	TEMP	CAL VF-Q =	GET VF-Q	CAL VFECON =	GET VFECON	CAL VFLIM =	GET VFLIM
3	----- Temp Up side Opration -----						
4	25.556	4	4	4	4	10.72222222	10.72
5	26.111	4	4	4	4	10.44444444	10.4
6	26.667	4	4	4	4	10.16666667	10.17
7	27.222	4	4	4	4	9.888888889	9.89
8	27.778	4	4	4	4	9.611111111	9.61
9	28.333	4	4	4	4	9.333333333	9.33
10	28.889	4	4	4	4	9.055555556	9.06
11	29.444	4	4	4	4	9	9
12	30.000	4	4	4	4	9	9
13	30.556	5.09	5.09	4.09	4.09	9	9
14	31.111	5.46	5.46	4.46	4.46	9	9
15	31.667	5.87	5.87	4.87	4.87	9	9
16	32.222	6.3	6.3	5.3	5.3	9	9
17	32.778	6.3	6.3	5.3	5.3	9	9
18	33.333	11.5	11.5	10.5	10.5	9	9
19	33.333	11.5	11.5	10.5	10.5	9	9
20	33.333	11.5	11.5	10.5	10.5	9	9
21	33.333	11.5	11.5	10.5	10.5	9	9
22	33.333	11.5	11.5	10.5	10.5	9	9
23	33.333	11.5	11.5	10.5	10.5	9	9
24	----- Temp Down side Opration -----						
25	32.778	6.72	6.72	5.72	5.72	9	9
26	32.222	6.3	6.3	5.3	5.3	9	9
27	31.667	5.87	5.87	4.87	4.87	9	9
28	31.111	5.87	5.87	4.87	4.87	9	9
29	30.556	5.09	5.09	4.09	4.09	9	9
30	30.000	4.72	4.72	4	4	9	9
31	29.444	4.5	4.5	4	4	9	9

オラクルの結果とアプリの出力を比較

テストオラクルのメリット／デメリット

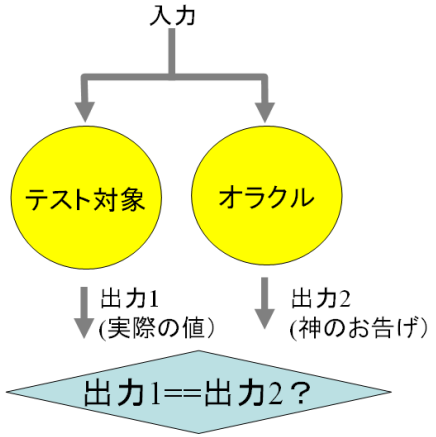
メリット

■ テスト設計の標準化。
「オラクル」「テストケース」の基本構成。

■ 仕様変更対応が容易。
オラクルを変更し、後は「入力条件」を考える。

■ 要件検証の強化。
オラクル作成時に要件の間違いに気づける。

■ 設計と平行してテスト設計可能。
やり直しテスト設計が減少。
（従来はいざ実行しようとしてもできないテストがあった）



デメリット

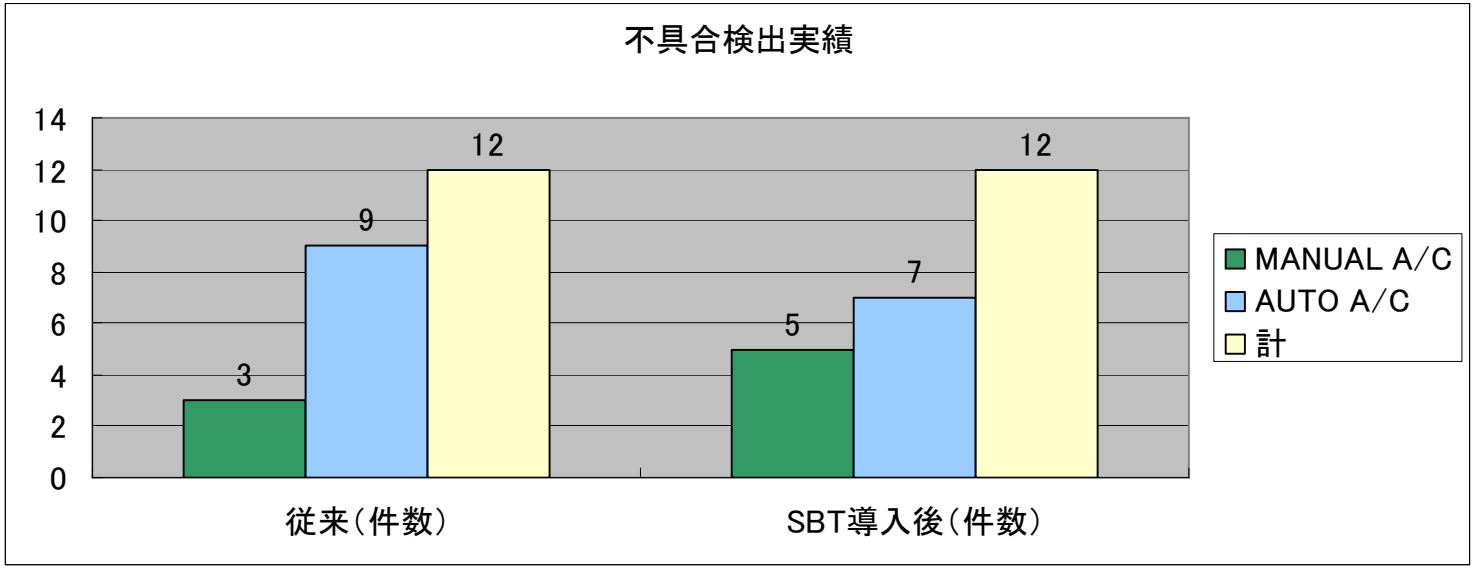
◆ オラクル作成には経験とスキルが必要。
教育体制の整備、新たな基準作成が必要。

Contents

品質面の効果

Q: 不具合検出実績

評価：★★★☆☆

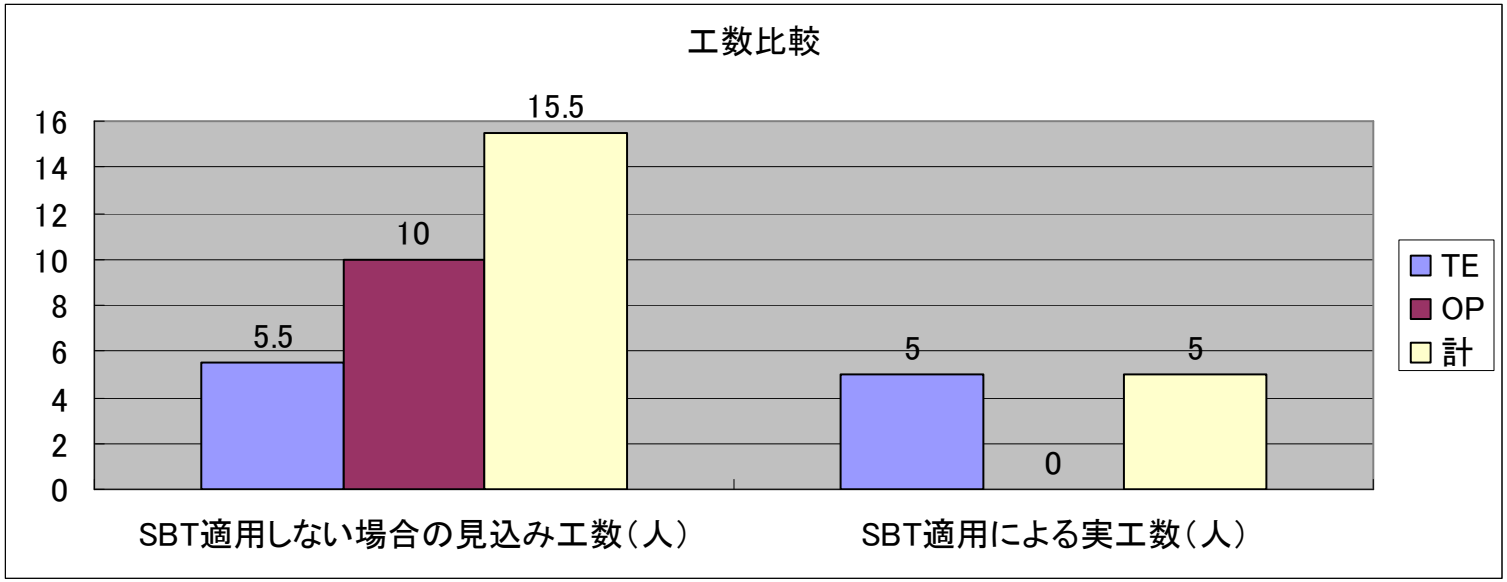


従来平均と同等レベル

効率化の効果

C: 作業工数

評価：★★★★★



TE: テストエンジニア

OP: オペレータ

従来より3倍以上効率化

楽しさの効果（やりがい）

Pleasant: 楽しさ　　評価：★★★★★

■SBT実施者の感想

1. テストシナリオを考える楽しさ
   C#コードで作成するテストシナリオは「発展性」を持っている。
   「網羅性重視」「効率性重視」「タフネス重視」「プレゼンテーション重視」
   いろいろなテストシナリオを考え、展開することができる。

2. シミュレータを考える楽しさ
   「機材が無くてテスト出来ない」「動作が理解し難い」など、シミュレータの
   改造・展開で解決することが可能。
   テストエンジニアとしての腕の見せ所。

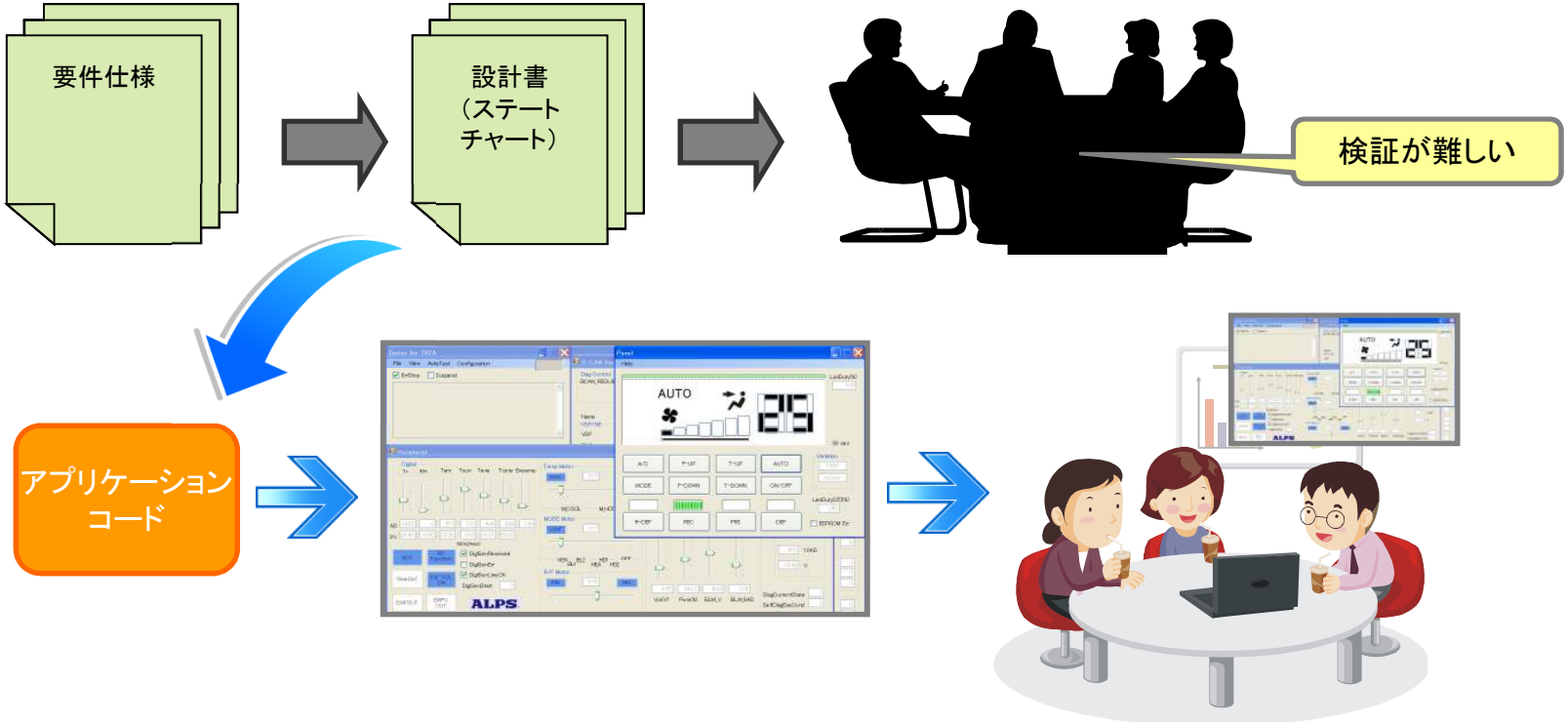
Contents

今後の展開　(1/3)

シミュレータ活用による要件検証の推進

　シミュレータには客先要件の殆どを実装したアプリケーションをそのまま組み込んでいるため、顧客との動作確認に使用可能。

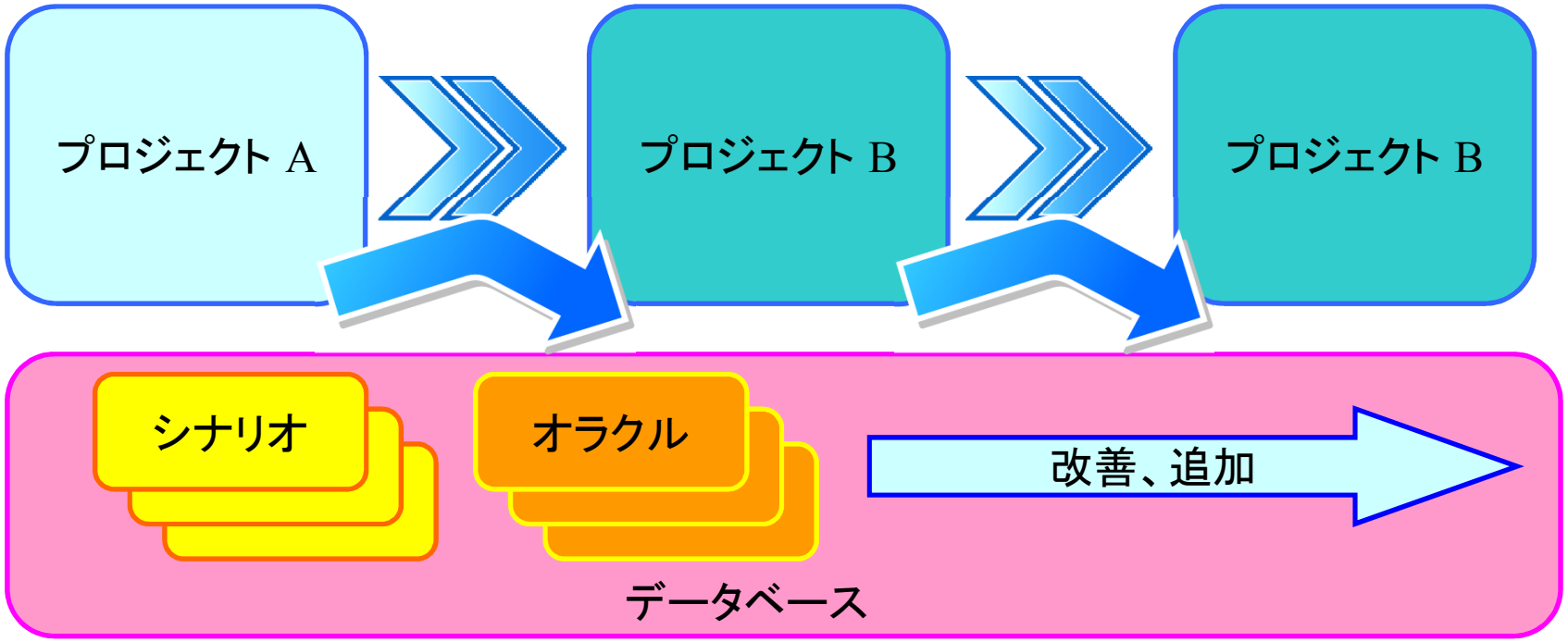
　文章ではなかなか分かりにくい機能も、シミュレーションで確認することにより精度の高い検証が可能に。



今後の展開　(2/3)

テストシナリオ再利用の促進

再利用性を考慮したクラス分けや操作分けを行い、そのまま使える資産としてテストシナリオを蓄積 。



今後の展開　(3/3)

オラクルと自動テストを活用してマニュアルテストを超える圧倒的なバグ検出率を目指す。

具体例:

入力をランダムに変化させてテスト実行の度に違う入力でテストできるようにする。

このテストを毎晩実行しテスト技術者が寝てる間にバグを見つける。

おわり

ご清聴、ありがとうございました。