原点回帰！
作業のしやすさに着目した
単体試験方法の提案
～単体試験時に用いるチェックシートの紹介～

（株）ニコンイメージングシステムズ
清水 崇司

2012年10月11日

本日の内容

- 会社、職場、開発体制の紹介
- 動的テストツール見直しの経緯
- 従来の単体テスト方法の問題点
- 新たな取り組み　～チェックシートの作成～
- チェックシートの効果、課題
- この取り組みのまとめ
- 最後に

あらすじ

「開発者」であり「SEPG」でもある発表者が

「動的テストツール」は「作業しずらい」と思い、

「SEPG」として「作業しやすさ」に着目し、

「チェックリスト」を提案した。

（「動的テストツール」の「作業しやすさ」の改善も一つの方法だが、「チェックリスト」によるチェックに「原点回帰」することで）

開発部内で一定の効果を得た。

会社、職場の紹介

- 会社名：株式会社ニコンイメージングシステムズ
- 設立：2009年7月1日
- 所在地：東京都品川区西大井1-6-3
- 資本金：9,500万円
- 出資比率：ニコンシステム 70%、富士通BSC 30%
- 従業員数：約100名（2012年9月1日時点）
- 事業内容：ニコン製デジタルカメラおよび交換レンズ用ファームウェアの開発

開発体制

組織：1グループ10名前後のグループが数グループで
　構成されている開発組織。（この発表で紹介する組織。）

開発形態：　派生開発・流用開発

開発対象：　組み込みファームウェア

担当工程：　開発者が設計からテストまで行う

発表者の担当：開発者であり、現場SEPG

「単体テスト」方法の見直しの経緯

【開発現場への要求事項】

・短納期化　　・コストダウン

→品質維持して効率を上げないと達成できない

開発者が**やりずらい**と感じていた
「動的テストツールでの単体テスト」について、
代替の方法で品質維持する方法を模索した。

動的テストツールについて

●数年前 「単体テスト網羅率」の目標設定
→開発部内で動的テストツール（動的ツール）の使用が推進
・テストの網羅率の測定のため、テスト作業の自動化のため

その後、プロジェクトの目標からテスト網羅率についての目標が無くなっても、動的ツールを使用して試験結果を残していた。

【動的ツールの使い方】
- 動的ツールで関数毎の入力値の組み合わせを出力する。
その入力値を用いて、動的ツールを動作させると、関数毎に、入力値を処理した出力値が出力される。
開発者は、この値が正しいかを確認する。

動的ツール使用上の問題

発生した状況	問題点
試験が始められない。	環境整備に手惑う。 H/W毎にツールのカスタマイズ作業が必要。
	ライセンス数が少ない。
試験に多くの時間がかかる。	実行中にエラーで止まる。
	テストケースの増大。
	テストの設定から、完了するまでの時間が長い。
動的ツールの使用レベルが開発者毎で異なる。	ツールの教育が不十分。
欠陥が見つからない。	テストは動的ツールが行うもの、と錯覚しやすい。

作業しずらい！

・動的ツールを正しく使用できていない状況だった。

原点回帰

【欠陥を見つけるために必要なことは？】

・欠陥は開発者しか判断できない。
　動的ツールが〇/×と示しても、それを確認するのは開発者。

・開発者が手間をかけないと欠陥は見つからない。

- **ツールを使うための作業に多くの工数をかけるより、1つでも多くの観点を持ってソースコードを確認することが重要では？**

→動的ツールを使用しないで単体テストを行う方法を検討した。

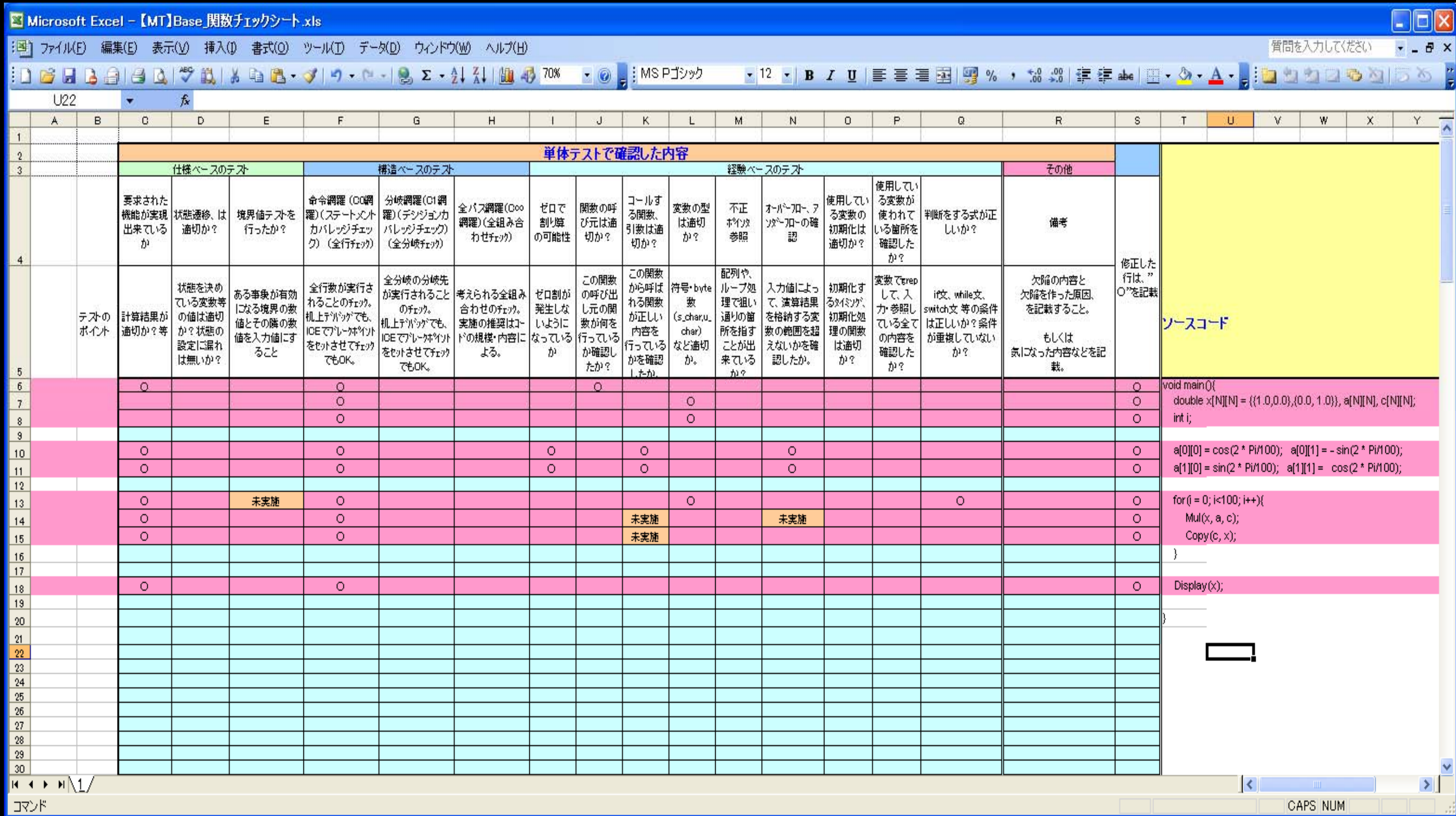
チェックシートの作成

【チェックシート作成で留意したこと】

ねらい	主な方法
単体テストを 「作業しやすい」作業にする。	1つの表にソースコードと チェック項目をまとめる。
試験項目のレベルを維持するようにする。項目漏れが起きないようにする。	試験の「観点」をリスト化する。
試験項目のレベルのボトムアップ。	過去に発生した欠陥から確認項目を作成する。 ベテラン技術者の確認項目を盛り込む。

チェックシートの紹介

・コードレビューの際に、試験項目をレビューする。



Microsoft Excel - 【MT】Base_関数チェックシート.xls

	単体テストで確認した内容																	
	仕様ベースのテスト			構造ベースのテスト			経験ベースのテスト									その他		
	要求された機能が実現出来ているか	状態遷移、は適切か？	境界値テストを行ったか？	命令網羅（C0網羅）（ステートメントカバレッジチェック）（全行ﾁｪｯｸ）	分岐網羅(C1網羅)（デシジョンカバレッジチェック）（全分岐ﾁｪｯｸ）	全パス網羅(C∞網羅)（全組み合わせﾁｪｯｸ）	ゼロで割り算の可能性	関数の呼び元は適切か？	コールする関数、引数は適切か？	変数の型は適切か？	不正ﾎﾟｲﾝﾀ参照	ｵｰﾊﾞｰﾌﾛｰ、ｱﾝﾀﾞｰﾌﾛｰの確認	使用している変数の初期化は適切か？	使用している変数が使われている箇所を確認したか？	判断をする式が正しいか？	備考		
テストのポイント	計算結果が適切か？等	状態を決めている変数等の値は適切か？状態の設定に漏れは無いか？	ある事象が有効になる境界の数値とその隣の数値を入力値にすること	全行数が実行されることのﾁｪｯｸ。机上ﾃﾞﾊﾞｯｸﾞでも、ICEでﾌﾞﾚｰｸﾎﾟｲﾝﾄをｾｯﾄさせてﾁｪｯｸでもOK。	全分岐の分岐先が実行されることのﾁｪｯｸ。机上ﾃﾞﾊﾞｯｸﾞでも、ICEでﾌﾞﾚｰｸﾎﾟｲﾝﾄをｾｯﾄさせてﾁｪｯｸでもOK。	考えられる全組み合わせのﾁｪｯｸ。実施の推奨はｺｰﾄﾞの規模・内容による。	ゼロ割が発生しないようになっているか	この関数の呼び出し元の関数が何を行っているか確認したか？	この関数から呼ばれる関数が正しい内容を行っているかを確認したか。	符号・byte数(s_char,u_char)など適切か。	配列や、ループ処理で狙い通りの箇所を指すことが出来ているか？	入力値によって、演算結果を格納する変数の範囲を超えないかを確認したか。	初期化するﾀｲﾐﾝｸﾞ、初期化処理の関数は適切か？	変数でgrepして、入力・参照している全ての内容を確認したか？	if文、while文、switch文 等の条件は正しいか？条件が重複していないか？	欠陥の内容と欠陥を作った原因、を記載すること。 もしくは 気になった内容などを記載。	修正した行は、"O"を記載	ソースコード
	O			O				O									O	void main(){
				O						O							O	double x[N][N] = {{1.0,0.0},{0.0, 1.0}}, a[N][N], c[N][N];
				O						O							O	int i;
	O			O			O		O			O					O	a[0][0] = cos(2 * Pi/100); a[0][1] = - sin(2 * Pi/100);
	O			O			O		O			O					O	a[1][0] = sin(2 * Pi/100); a[1][1] = cos(2 * Pi/100);
	O		未実施	O						O					O		O	for(i = 0; i<100; i++){
	O			O					未実施			未実施					O	Mul(x, a, c);
	O			O					未実施								O	Copy(c, x);
																		}
	O			O													O	Display(x);
																		}

チェックシートの特徴

従来の問題点	チェックシートでの試験の特徴
環境整備に手惑う。H/W毎にツールのカスタマイズ作業が必要。	環境整備なし。 （試験項目の改善は）
ライセンス数が少ない。	関係なし。
実行中にエラーで止まる。	なし。
テストケースの増大。	なし。 ※品質に影響する可能性あり
テストの設定から、完了するまでの時間が長い。	テストはすぐに開始できる。 実施の際に待たされることがない。
ツールの教育が不十分。	チェックシート使用方法の教育はわずかで済む。
テストは動的ツールが行うもの、と錯覚しやすい。	テストは開発者が記録しないと終わらない。錯覚はしない。

チェックシートの効果、課題

【効果】

- 検出率の向上（変更行数に対する検出欠陥数の向上。）
- 試験効率の向上
- 試験しやすい環境になった。（作業者のイライラ減少）
- チェック項目（観点）が明示されているため忘れない/漏れにくい
- 試験項目数の計上も行える。
- ツールの教育がいらなくなった。
  欠陥を見つけるためにはどうすればいいかを考える機会を作れた。

【課題・取り組みたいこと】

- 試験項目の粒度の整理。網羅性の試験の質の向上。
- 個人レベルでカスタマイズしたい。
  （PSPのコードレビューチェックリストのように。）

動的ツールの使用について

●テストケースを作成して使いまわす（夜間運転などで自動化する）ことでコストダウンできる、 という言葉を聞くけれど、、。

①あるソフトウェアのテストケースは一度テストで使ってしまえば、同じソースコード、同じテストケースでは、基本的に新しい欠陥は見つからないのでは？（関数レベルでの複合的なテストケースが確立されていれば意味があるかも？）

②ソースコードを変更したら、テストケースを作成しなおしてテストしなければならない。影響範囲を正確に把握していれば動的ツールは必要ない？

安直に、**動的ツールを使えば自動化できて、コストダウンできる**と考えてしまうときは、ご注意を。

今回の取り組みのまとめ

●作業しやすい環境は、
　無駄な時間を減らし、作業の質を向上させる。

●欠陥を見つけるために必要なことは？

最も重要なのは、
欠陥を見つけるためのめんどくさい作業に
ネガティブな印象を抱かないようにすること
だと考える。
　このため、作業しやすい環境の整備は不可欠。

最後に

開発者である私たち：

作業しずらいと感じることがあったら、
声に出していきましょう。
ソフトウェアの品質を作り上げるのは私たちです。

SEPGである私たち：

合理性を「作業のしやすさ」に求めてみませんか？
開発者とSEPGがWIN-WINになれた事例があります。
ソフトウェアの品質を作り上げる組織を作るのは私たちです。


Nikon