SPC 研究会第 5 分科会(B グループ)

単体テスト実行による品質向上(短納期対応)

Quality improvement by unit test execution (short delivery response)

分科会メンバー

主　査	保田　勝通	(つくば国際大学)
副主査	高橋　寿一	(ソニー株式会社)
	西　　康晴	(電気通信大学)
研究員	小笠原健二	(日立システムアンドサービス株式会社)
	小越　興一	(セントラル・コンピュータ・サービス株式会社)
	越野　僚太	(三菱電機コントロールソフトウェア株式会社)
	小林　利治	(日本電子株式会社)
	竹本　昇司	(株式会社野村総合研究所)

概要

近年のソフトウェア開発における課題の一つとして、短納期開発が挙げられる。
以前に比べ短くなる開発期間のなかで、いかに効率良く品質を作りこむかが大きな課題となる。テスト工程のなかでも、上流工程での品質確保として「単体テスト」の重要性は理解されていても、進捗の遅れや環境構築などの問題で疎かになったり、実施されなかったりする場合が多くなっている。そのため、「結合テスト」や「システムテスト」また本稼働において、「単体テスト」で摘出すべき不具合が発生している。

本研究では、「単体テスト」における問題点を洗い出し、必要性，位置付けを明確にして、成功事例や実際のテスト実施例で効率化や品質向上のメリット，デメリットをまとめて、単体テストの重要性について検証を行う。

Abstract:

The short delivery date development is enumerated as one of the problems in the software development in recent years. How bulid the quality very efficiently become big problems in the development period that shortens compared with before.

The "Unit test" often becomes sparse or doesn’t execute due to the delay of progress and the problem of environmental construction etc. ,even if understood the importance of the “Unit test” as the quality securing in the upstream process in the test process.Therefore, the defect that should be removed by "Unit test" occurs in "Combined test", "System test", and real operation.

In this research, the importance of the unit test is verified digging up the problem in "Unit test", clarifying the necessity and the location, and bringing the advantage and the disadvantage of efficiency improvement and the quality improvement together by the example of executing a success case and an actual test.

1 章　はじめに

1.1 研究の目的

ソフトウェア開発において、単体テストを実施することにより、どのように品質向上、短納期対応に効果があるかを検証する。

1.2 論文の構成

下記の構成とする。

1.はじめに(本章)
2.単体テストの必要性、位置づけ
3.現状の単体テストの問題点
4.システムテストから見た単体テストの位置づけ
5.単体テスト実施による成功事例
6.単体テスト実施事例
7.まとめ
8.今後の課題

2 章　単体テストの必要性、位置付け

納品物が品質要求事項を満たすために我々が実施すべき品質保証に関する具体的な活動は、以下の３つに大別される。

（１）予防
　品質不良の予防を目的とする。例えば、文書化標準、手法、技法など
（２）検知
　品質不良の検知を目的とする。例えば、レビューやウォークスルー、テストなど
（３）修正
　品質不良を修正することを目的とする。例えば、バグの修正など

品質保証活動においてテストは検知における１つの活動と位置付けられる。経験的には、検知にかける期間が短いと不具合の発見が不十分となり、その結果不具合が残る。残った不具合による障害が発生した場合、障害に対応するための工数がかかる。また、不具合の発見が遅れるにつれて、不具合の修正のためにかかる工数は増える。我々は、予防、検知及び修正といった品質保証活動を効果的に実施する必要がある。テストを適切に実施することは、品質保証活動に費やすコストを最適化する上で重要と言える。

ソフトウェア開発ライフサイクルの各工程における品質保証活動とその役割について、表 2-１にまとめる。

表 2-１ ソフトウェア開発ライフサイクルの各工程における品質保証活動とその役割

工　　程	品質保証活動			
	設計工程	目　　的	テスト工程	目　　的
要求分析	要求事項レビュー	要求事項の漏れがないかを確認する	システムテスト	システムが要求事項を満たすよう動作することを確認する
基本設計	基本設計書レビュー	機能の漏れがないかを確認する	結合テスト	ソフトウェアが設計書中の機能要求を満たすよう動作することを確認する モジュール間のインターフェイス中心に確認を行う
詳細設計	詳細設計書レビュー	モジュール単位で設計漏れがないかを確認する	単体テスト	プログラムが設計書中の要求を満たすように動作することを確認する モジュール単位の Input/Output 中心に確認を行う(限界値、境界値、エラー処理)
コード作成	コードレビュー	プログラムが設計書中の要求を満たすように記述されていることを確認する		

表 2-１より、実際の動作の確認はテスト工程においてのみ確認できることがわかる。その中で単体テストは一番上流の工程であり、細分化された範囲のテストと位置づけられている。

3 章 単体テストの抱える問題点

前章で見たように、単体テストは開発の工程として一般的に認識されている。それにもかかわらず、実際の開発においては、短納期、前工程の遅れによるしわ寄せはテスト工程に影響をおよぼす。テスト工程のなかでも特に単体テストは軽視、あるいは省略されることが多い。

分析の結果、この傾向は単体テストの抱える問題によることがわかった。単体テストが抱える問題を表 3-1 に示す。

表 3-1 単体テストの抱える問題

	問題	問題の詳細
1	デバッガ環境でのシミュレータなどの環境作りが大変	組み込みシステムの単体テストではプログラム単体でのテストを行うため、実機とは切離されたデバッガ環境でテストを行うことになる。デバッガ環境の利用においては下記の問題がある。 ・開発ソフトウェアと連携して動くソフトウェアシミュレータを用意する必要がある ・開発プログラムを動かすためのテストデータを用意する必要がある
2	モディファイ開発の場合、プログラムコード全体の網羅的なテストは行わない	組込みシステムの場合、モディファイでの開発が多い。モディファイ元の既存プログラムにおいては、すでに稼動の実績がある。そのため、モディファイ部分中心のテストとなり、網羅的な単体テストは行わないことが多い そのため、単体テストの対象範囲が自分の関心のある部分中心となり、新規開発でも全体のテストを行わなくなる傾向が出やすい。
3	単体テストの記録が残らない	結合、システムテストでは、組織としてテストを行うため、記録を残し、分析、改善を行う。対して、単体テストではそれぞれの開発者でのテストとして、統一的な管理がされていないケースもある。その場合、詳細な記録は参照されないことが多く、分析することはあまりしないので、記録を残すという意識が薄い。
4	結合テストの実施を急ぎたくなる	動作にかかわる重大な不具合は基本設計や、それ以前で発生することが多いが、前章でみたように、基本設計で発生した不具合は単体テストでは発覚せず、結合テスト以降で発生することになる。 また、実際のプログラムの動作、あるいは不具合は実機と組み合わせないと目に見えないため、短工期になるほど結果を早く目でみたくなり、単体テストを省略して結合テストに入ってしまう傾向がある。

4章　システムテストから見た単体テストの位置付け

システムテストではシステム全体の動作を通して機能要件や非機能要件の要求事項を満たすかどうかを検証する。

システムテストの実施においては、その前段となる「単体テスト」「結合テスト」が実施され大半の不具合が発見され対応が取られていることが前提となるが、実際の開発においてはシステムテストで発見される不具合で手戻りが発生し、結果として工期の圧迫に繋がっている。

次に挙げる表は、比較的小さなプロジェクトにおけるシステムテストにおいて発見された不具合の管理表である。

このプロジェクト事例では、「単体テスト」「結合テスト」はコード実装者が実装作業中に行なっている。

表 4-1 システムテストにおいて発見された不具合管理表

現象	部位	数	原因	発見されるべきテストフェーズ
制御論理に不適合処理	GUI コントロール部品	5	関数呼び出しの引数理解不足	結合テスト
表示文字列に不適合文字	GUI 表示部	7	文字綴り間違い	単体テスト
ハードウエアへのアクセスが遅い	全体	1	ネットワーク遅延	システムテスト
制御挙動が不確定	関数モジュール	3	関数モジュール内変数初期化が不適格	単体テスト
メモリアクセス違反	関数モジュール	1	文字列配列への不適格コピー	単体テスト
メモリリーク	関数モジュール	1	獲得メモリの開放漏れ	単体テスト

(1)単体テストで見つけるべき不具合が残ってしまったケース

これら不具合の中で修正に、もっとも工数を要したものは「メモリアクセス違反」であった。

アクセスするメモリ領域はシステム内の他モジュールからもアクセスする領域であったため、当該モジュールのメモリアクセス違反を修正した際に、関連他モジュールのメモリアクセスロジックも確認、一部モジュールについては修正が必要となった。その結果、結合テスト、システムテストのかなりの部分のやり直しが必要となった。

この不具合個所の特定には、デバッガを使い、操作シーケンスをステップ実行でトレースし不具合個所を含む関数モジュールを発見するという、通常の単体テストの手順で調査を行った。単体テスト実施時に十分発見できるものである。また、単体テスト時に当該不具合が発見できていれば、関連モジュール開発者に情報連携でき、修正の必要なモジュールが特定でき、結合テスト以前に問題は解消できていたはずである。

「メモリアクセス違反」のような不具合は個別モジュールの単体テストで見つけやすい不具合である。共通的な不具合がシステムテストまで持ち越されると、上記のように多大な手戻りが発生する。

(2)単体テスト実施記録がシステムテストでの不具合調査に役に立ったケース

一方、ＧＵＩ表示や制御論理に関する不適合の調査、修正では単体テストの実施記録が役立っ

た。

テスト者から実装者へ不適合の現象が報告された際に、実装者では実装時におこなった単体テストの実施記録から、各モジュールの境界値に関する実装仕様を明確にすることができた。その結果、仕様解釈の齟齬が発見され、不適合の原因が解明でき、修正箇所、影響範囲の絞込みが容易になり、修正を迅速に行うことができた。

「単体テスト」を行い、実施結果を記録することで、システムの粒度の小さな部分の透明度を向上させ、システム全体の処理の見える化が行われた。これにより、不具合が発生した場合の障害個所の特定を迅速に行うことができた。

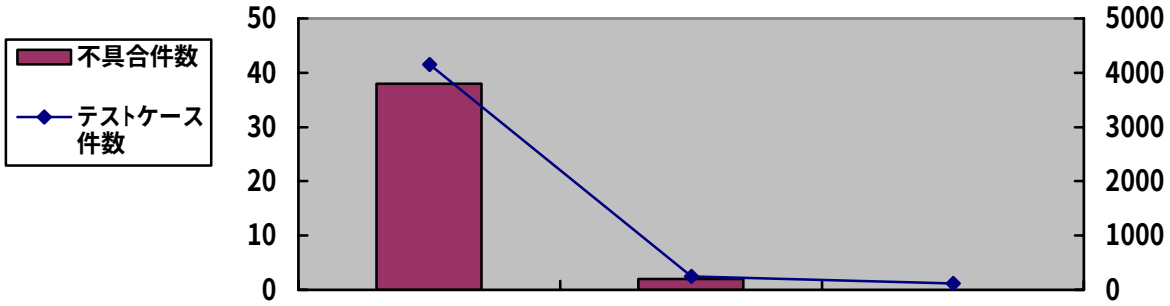
5 章　成功事例

単体テスト工程での不具合摘出率が高く、結果 品質向上や効率向上につながった成功事例として 2 つのプロジェクト事例を以下に示す。

【事例１：プロジェクト A】

使用言語：Ｃ＋＋　　開発規模：新規１２．１Ｋｓ

開発期間／人員：計画 ･･･ ５ヶ月／７人、実績 ･･･ ４．５ヶ月／７人



テスト工程	単体テスト	結合テスト	システムテスト
テストケース件数(件)	４１５０	２４２	１２０
不具合件数(件)	３８	2	0
不具合摘出率(件)	９５％	５％	０％

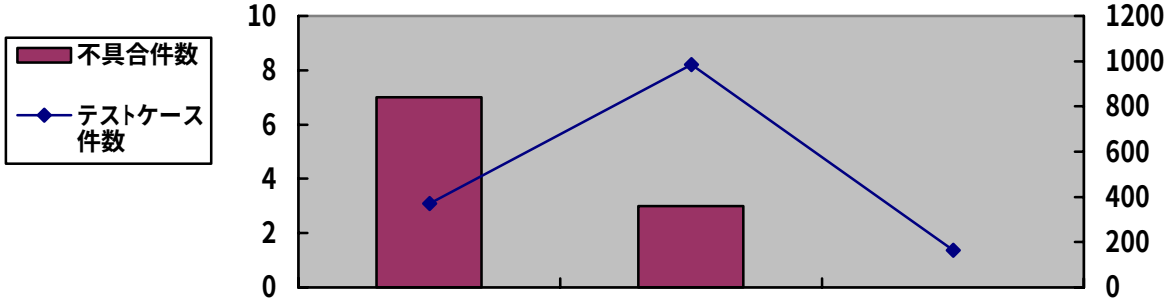
図 5-1 新規開発における各テスト工程での不良摘出率推移

＊新規開発であり、単体テストレベルを網羅的に実施することで、早い段階で不具合を摘出できている。

【事例２：プロジェクト B】

使用言語：ＶＣ＋＋　　開発規模：母体４６．５Ｋｓ、新規１．５Ｋｓ、改造０．５Ｋｓ

開発期間／人員：計画 ･･･ ２ヶ月／５人、実績 ･･･ １．８ヶ月／５人



テスト工程	単体テスト	結合テスト	システムテスト
テストケース件数	３７０	９８６	１６５
不具合件数	7	3	0
不具合摘出率	７０％	３０％	０％

図 5-2 改造開発における各テスト工程での不良摘出率推移

＊母体からの改造＋新規であり、単体テストでは新規開発部分を重点に実施することで、早い段階で不具合を摘出、かつ効率のよいテストができている。

いずれのプロジェクトにおいても、単体テスト工程での不具合摘出率が高く、早い段階での品質の作り込みができているものと考える。また、前工程での不具合摘出により、後戻りが少なくなった結果から、効率向上にも繋がっていると考える。

6 章　単体テスト実施事例

今まで単体テストをあまり実施せず、「結合テスト」や「システムテスト」また本稼働において、「単体テスト」で摘出すべき不具合が発生していた組織において、今回の開発プロジェクトでは実機テスト環境構築までに単体テストを実施した。その事例を以下に示す。

6.1 単体テスト実施にあたっての工夫点

単体テスト実施にあたり、以下の点において工夫を行った。

(1) 誰でも実施できるようＰＣ上で実施できるツール（デバッガ）を採用した。

(2) ファンクション（関数）単位の管理表を作成し、実行結果の記録を残した。

(3) ツールの有識者から講習を受け、取扱いマニュアルを標準化した。

(4) 不具合の分析と再発防止策検討、テストケースを上げる際の着眼点とするため、不具合内容を分類した。

6.2 単体テスト実施による品質・効率向上の結果

単体テストを実施した結果により、１０件の不具合を摘出し、早い段階での品質向上をすることができた。単体テストの実施方法、実施結果、メリット、デメリットを以下に示す。

(1) 実施方法

①ＰＣ上で実施するファンクション（関数）単位の単体デバッグで、スタブ、ドライバを用意し、環境構築を行う。

② ファンクション（関数）で必要なテストデータを作成／実行し、管理表に実行結果を記録する。

③ 単体テストは、今回改修を加えた箇所で、複雑な計算を実施している箇所と、正常パスが通ること(無限ループ等に陥っていないこと)に着目して実施した。

・異常ケースについてはシステムテストで実施するため、単体テストは実施せず。

・流用元から変更しなかったところは単体テスト実施せず。

・アルゴリズムが複雑な物に関しては、コードレビューにて検証した。

（1） 実施結果

品質向上において、１０件の不具合を摘出できた。摘出した不具合を表 6-1 に示す。

表 6-1　単体テスト実施により摘出した不具合

	摘出した不具合内容	件数(件)
1	if 文の判定条件が"=="のところが"="であった	2
2	初期化ミス	5
3	モジュールのコール忘れ	2
4	配列バッファ容量を超える添え字宣言	1

効率向上において、下記の通り向上効果があった。

①新規ハードウェアが来る前に不具合を洗い出せた。

②実機で不具合が出ても、単体テストで確認した箇所は問題ないと判断できた。

（2） メリット・デメリット

単体テストを実施した際に感じたメリット、デメリットを表6-2に示す。

表6-2　単体テストを実施した際のメリット・デメリット

	メリット	デメリット
品質面	新規ハードウェアが来る前に不具合を検出できた	なし
	テストケースを意識することで、プログラムの動作理解が深まった	
効率面	単体テストで用いたデバッガアプリケーションが実機デバッグ時のデバッガアプリケーションと同様のため、実機デバッグ時に抵抗なく、扱うことができた	スタブ、ドライバ等の設定に時間がかかった
	実機デバッグ時に単体試験を実施した箇所は正常とみなして実施できた	単体テスト実施記録を作成する手間がかかった
	ハードウェアとの組合わせ時に問題の絞込みが容易になった	やはりハードウェアがないとデバッグできないところが多かった(ファームウェア的な箇所)

6.3　単体テスト実施についての見解

上記組織で単体テストを実施した後の単体テストへの見解は以下のとおりであった。

(1)実機テスト前に不具合を検出することができた

(2)単体テストを実施したことで、実機テストの品質、効率が向上した

(3)単体テストを実施するにあたり、単体テスト環境、特にテストデータ投入を現状の実機テストなみに容易に実行できることが必要

(4) ファームウェアと連動するところなどは、単体テストでは確認できないため、実機にて確認する必要がある

７章　考察

7.1 まとめ

前章まで見てきたように、単体テストを実施することによって品質面は向上する。
一方、効率面については向上するものもあるが、負荷増大するものもある。以下にそれを示す。
(向上するもの)
①以後のテストでのテスト効率向上

- 単体テストレベルでの不具合が解消されているため、手戻りが少なくなる

②デバッグ効率(不良切り分け効率)向上

- 障害切り分けに単体テスト実施記録を利用することができる

(負荷増大するもの)
①単体テスト環境を構築する負荷増大

- テストツールを選定し、セットアップする必要がある
- ツールに投入できる形式でテストデータを用意する必要がある
- テストモジュールを実行するためのドライバ、スタブ(シミュレータ)を用意する必要がある

②単体テストの実施記録作成負荷増大

- 単体テストの実施記録を作成する必要がある

7.2 負荷増大要因対応策

短納期対応するためには、上記の負荷増大要因に対応する必要がある。以下にそれぞれの要因への対応策案を示す。
①単体テスト環境を構築する負荷増大

- 環境を継続して利用
- データを継続して利用

②単体テストの実施記録作成負荷増大

- 実施結果記録の標準化による負荷軽減
- 実施結果を自動記録できるようなツールの採用

８章　今後の課題

今回の研究では、ソフトウェア開発において、単体テストを実施することにより、品質向上、短納期対応に効果があることを検証したが、短納期に対応するための単体テスト効率化施策検討までは至らなかった。
今後一層の短納期対応のためには、単体テストの実施の効率化研究を行う必要がある。研究に当たっては下記項目を留意する必要がある。

① 利用しやすい単体テスト環境
② 単体テストの実施結果を容易に記録する手法
③ プロジェクトの性質に応じた適切なテストケースの設定

参考文献

テストプロセス改善-CMM 流実務モデル-」Tim　Koomen,Martin Pol、構造計画研究所、2002 年