事例紹介:OEMソフトウェア製品の検証プロセス

名倉　正剛†，田村　清朗†，川口　真司‡*，高田　眞吾☆，飯田　元‡

他組織がOEMによって開発したソフトウェアを販売する場合,販売を行う企業は自社製品と同様に品質に責任を持つ必要がある.しかし資本関係のない他組織が開発しているため,組織間で同程度の要求を満たすように設計されていないことが多い.そのため他組織の開発したソフトウェアを利用しても期待するほど生産性が高くならない.本研究では,海外他社がOEMによって開発した企業業務に利用されるソフトウェアを自社製品に導入する際に行った検証プロジェクトの経過観察をした.その結果,このプロジェクトでは重要な影響を与える不具合が,検証プロセスの後半に発見されていた.発見された不具合の報告状況から,そのような状況を回避するために,発注先で実施した単体テストや統合テストに関する検証項目のリストの開示,要求定義に関する追加的な情報の開示の効果を考察した.そして,それらを開示した場合の検証作業プロセスに関する指針を述べた.

Case Study : Verification Process for an OEM Software Product

Masataka Nagura† ,Seiro Tamura† ,Shinji Kawaguchi‡* ,Shingo Takada☆, and Hajimu Iida ‡

When a company sells a product made by another company as an OEM product, that company is responsible for the quality of the OEM product in the same way as its own product. However, because the two companies do not have any ties in terms of capital, the OEM product is often not developed according to the shared requirements of the two companies. Accordingly, the productivity due to incorporating OEM is not as high as expected. In this paper, we describe the results of observing the verification process that was taken for an actual OEM software product. We especially found that critical bugs were found later in the verification process, which could be avoided if information such as the original requirements and test data were made available.

1 はじめに

ソフトウェア製品やその開発プロジェクトは，年々大規模になっている.そのため,ソフトウェア開発組織間でモジュールごとに受発注を行ったり，更には他社がOEM（Original Equipment Manufacturer）として開発したソフトウェアを自社製品として導入したりすることによって，開発コストを削減した上で大きな利益を得られるように，開発体制が変化している.

一般的に商用ベースのソフトウェア開発において，他組織が開発したソフトウェアを利用する場合，販売企業はその品質にも責任を持つ必要がある．しかし資本関係の無い他組織のソフトウェアを利用する場合，組織間で同程度の要求を満たすように設計されていないことが多い［BARBOSA2007］.このため，販売企業は他社開発のソフトウェア部品を利用する際に，再度検証を実施することになる．要求に関する認識が異なっていたり，要求自体が異なっていたりすると，この検証作業に多くのコストを要する．結果として他組織の開発したソフトウェアを利用しても，期待するほど生産性が高くならないというのが現状である.

本研究では，このような状況における検証の実例として，企業業務に利用される海外他社によってOEMによって開発されたソフトウェアを自社製品として導入する際に，実際に行われた検証プロジェクトの経過を観察した．そしてOEMソフトウェア製品を検証する際に，どのような問題が発生するか分析した．その上で，発注先と発注元企業でどのような情報をやり取りして，どのように検証プロセスを進めれば効率的に検証を実施出来るか，指針を示す.

† 株式会社日立製作所，Hitachi,Ltd.
‡ 奈良先端科学技術大学院大学，Nara Institute of Science and Technology
☆ 慶應義塾大学，Keio University
* 現在，有人宇宙システム株式会社

2 他組織の開発したソフトウェアを導入する際の課題

Robinson らは，複数の組織によってシステムを開発する際の発注元と発注先の関係を，ベンダ間の資本的関係と地理的関係によって，次のように分類している［ROBINSON2004］.

●資本的関係の無いベンダへ発注（Outsourcing）

① 国内ベンダに発注（Onshore Outsourcing）

② 国外ベンダに発注（Offshore Outsourcing）

●資本的関係のある同一組織内のベンダ（子会社，関連会社など）へ発注（Insourcing）

③ 国内ベンダに発注（Internal Domestic Supply）

④ 国外ベンダに発注（Offshore Insourcing）

このように，外部組織で開発したソフトウェア製品を利用してシステムを開発することを“Outsource”と定義し，関連会社などのような自組織内で開発したソフトウェア製品を利用してシステムを開発することを“Insource”と定義している．また，自国内での生産かどうかによって，“Onshore”及び“Offshore”に分類している．本論文における本節での記述では，Offshore Insourcing と Offshore Outsourcing を合わせて，「オフショア開発」と記載する．

ソフトウェアの大規模化により，近年では Outsource によって外部の組織が開発したソフトウェアを部品として利用してソフトウェアを開発する場面が多い．更には開発コストを削減した上で大きな利益を得られるように OEM によって開発された他社製ソフトウェアを自社製品に導入して販売することもある．このことは，主に開発期間の短縮や，人的／経済的な開発コストの削減を目的としており，新技術に追随するために OEM によって開発された海外製のソフトウェアを導入することもある．しかし実際には，Outsource 開発がコスト削減に必ずしも役立っていない場合も多い．

総務省がソフトウェア開発を行う日米の約 700 社へ実施したアンケート調査［MIC2007］において，オフショア開発を実施していると答えた 212 社の回答（日本 96 社，米国 106 社）を参照すると，オフショア開発導入時のコスト削減見込みに比べ，実際に得られた削減効果は 10% 近く低くなっている※1．なお，回答した企業には Insource 開発（［ROBINSON2004］における④に相当）を実施している企業も含まれていたが，多くは Outsource 開発（［ROBINSON2004］における②に相当）を実施していた※2．この結果から，海外の他組織で開発したソフトウェア製品を利用することが，期待するほどのコスト削減に結びついていないことが分かる．また，［ITO2010］では，それぞれの開発形態における生産性を比較している．これはソフトウェアに限らず日本の製造業の全業種を対象に生産性の推移を分析しているものであり，必ずしもソフトウェア開発の傾向を表しているとは言えないが，資本関係を持たない企業に対する Offshore Outsourcing（［ROBINSON2004］における②に相当）が生産性の向上に有意な効果を持たないことが報告されている．このように，［MIC2007］［ITO2010］のどちらからも，Outsource 開発がコスト削減に必ずしも役立っていないことが分かる．

Outsource によって外部の組織が開発したソフトウェアを部品として購入し利用する場合には，通常は次のような特徴が存在する［VIDGER1996］.

A） 発注元企業では，ソースコードにアクセス出来ない．また，内部文書が存在しない場合が多く，存在しても発注元企業からアクセス出来ない．

B） 技術情報（制限事項，パフォーマンス，リソース消費量等）は十分に記述されていない場合が多い．ユーザレベルの文書やトレーニング情報は，十分に記述されていることが多い．

発注元企業は，このような特徴を持ったソフトウェア部品を組み合わせてソフトウェアを開発する．Insource によって開発を実施する場合に比べ，開発プロジェクトを推進する際に参照出来る情報が大幅に少ない．なおこのような特徴は発注元と発注先との契約関係に依存するため，必ずしもすべてのプロジェクトに当てはまらない．しかし，OEM による Outsource 開発を実施する場合は，製品を納入することのみが契約になっていることが多く，このような特徴がとくに現れやすい．

他社開発のソフトウェア部品を導入する際には，発注元企業においても検証を実施する場合が多い．なぜなら，発注元企業では購入したソフトウェア部品に対して自社開発製品と同様の信頼性と品質を求めているものの，発注先においては発注元と同程度の品質を満たすような要求に基づいて設計されていない場合が多いからである［BARBOSA2007］※3．OEM によって開発されたソフトウェアを自社製品に導入する場合，既に販売されている製品が納入されるため，発注元では厳密な検証作業を省略出来るはずである．しかし導入したソフトウェアに対しても，発注元企業である自社がその品質に責任を持つ必要があるため，一般的には発注元企業の品質を満たすことを納入時に確認することが多い．このため，納入時に検証作業を省略出来ないことになる．このことが期待するほど生産性を高めることが出来ない一因になっていると考えられる．

我々はこのような状況を確認するため，実際のソフトウェア開発プロジェクトにおいて，導入時の検証作業としてどのようなことを実施しているかを観察した．本論文ではその一例として，海外ベンダが開発した OEM ソフトウェアを，自社製品群のラインナップに追加して販売する際に実施した検証プロジェクトの経過を紹介する．そして海外他社製品を導入する際の検証作業を推進する上で，どのような部分が問題になり検証コストが増加しているかを分析する．その上で問題を回避するために検証作業で留意すべき点について論じる．

脚注

※1 日本企業の場合，オフショア開発導入時にコスト削減効果は平均 34.7% と見込まれていたが，実績は平均 25.2% であったと報告されている．

※2 日本企業の場合，回答企業の 70.8% が，関連会社以外に開発を委託していた（［ROBINSON2004］における②に相当）．

※3 ［BARBOSA2007］では，入力に対して応答を返すタイミングや，ソフトウェア障害発生時のシステムに与えるインパクトに関する要求が挙げられている．

表1　検証対象ソフトウェアの規模

ソフトウェア全体		JAR ライブラリ		スクリプトファイル	
ファイル数	サイズ	ファイル数	サイズ	ファイル数	コード行数
20,917	1,442.9MB	932	732.1MB	385	92,733 行

表2　不具合報告のスケジュール

指摘回	経過期間	指摘内容に含める不具合についての初期方針	作業開始時の想定	
			重要度	報告数
第一回	約 12 週後	ユーザが必ず実施する手順に関連する不具合. （インストールや起動に関して発生した不具合や，当該ソフトウェアを利用する上で必ず行われる基本的な操作を行った際に発生する不具合など）	高	多
第二回	約 22 週後	場合によっては実施する可能性が高い手順に関連する不具合. （他製品と組み合わせる際の手順や，より専門的な作業を実施する場合に利用する当該ソフトウェアの詳細な利用手順に発生する不具合など）	中〜低	中
第三回	約 26 週後	上記に含まれない／上記において指摘漏れとして残った不具合で，リリースまでに緊急に修正する必要がある不具合. （リリース直前であるので，ドキュメントを添付することにより，ユーザに回避策を提示出来る場合は，この段階では不具合を報告しない）	高 （修正に多くの手間を要しないもの）	少

3 検証対象プロジェクト概要

本章では分析を行った検証プロジェクトの概要について述べる.

3.1 検証対象ソフトウェアの規模

検証対象ソフトウェアは主に Java で実装されており，一部についてはユーザのカスタマイズを可能にするために，スクリプト言語で実装されている．スクリプトを除いた部分の製品は，バイナリで納入されている．検証対象ソフトウェアの規模として，ソフトウェア全体のファイル数とサイズ，Java の JAR ライブラリファイル数とサイズ，スクリプトのファイル数とコード行数を表 1 に示す.

また，このソフトウェアにはユーザ向けのドキュメントが添付されていた．これは，2,221 ページから構成され，ほとんどの記載内容は，インストールと，UI 操作に関する説明であった. UI 操作に関する説明は，画面上の要素の操作（ボタンのクリック，テキスト入力など）に関してのみで，どのような場合にその操作を実行するのか，という利用シナリオに関する説明がほとんど記述されていなかった.

3.2 作業スケジュール

OEM によって開発された他社製品を自社製品として販売するにあたって，約半年間で検証作業を行った．社内製品を検証する場合とは異なり，検証した結果生じた不具合に関する指摘事項を随時報告出来る状況ではなかった．そのため，ある程度指摘事項を蓄積しておき，不具合による影響の重要度に応じ 3 回（約 12 週／ 22 週／ 26 週後）に分けて指摘を行うように，スケジュールを作成した（表 2）.

3.3 作業メンバの構成

検証作業は，合計 11 名のメンバによって行った．このうち 5 名は，10 年以上のソフトウェア開発経験を有しており，残りの 6 名は 1 年から 6 年であった．なお，メンバ全員が検証対象のソフトウェアの利用経験がない状態であった．検証作業の全期間に従事出来たメンバは 2 名のみで，他のメンバは途中から参加したり，途中で検証作業から抜けたりしており，全体では 40 人月の工数で検証を行った.

3.4 検証作業実施方針

今回の検証作業は他社が OEM によって開発したソフトウェアの自社製品への導入のための作業であり，ソフトウェアのどのモジュールがどのような目的に利用され，そのためにどのように操作すべきか，ということを把握出来ている開発者が社内にいない．そこで検証作業の初期段階では，動作の把握が急務であった

しかし 3.1 節で述べたように，ドキュメントのほとんどの部分は UI 操作に関する説明であり，ソフトウェアの動作を把握することは困難であった.

そのため，最初に次の作業の実施を計画した.

【フェーズ 1：UI の要素に基づく検証】ドキュメントに記載された各 UI の要素につき，チェックリストを作成した上で，動作の検証を行う．主に，UI の要素を操作したときの処理や画面遷移が，ドキュメントの記載通りに行われるかどうかを確認する．また，チェックリスト作成の際に UI の各要素を操作することにより，検証対象ソフトウェアの動作を把握する.

検証対象ソフトウェアの処理が実際に正しく動作しているかどうかについては，個々のUIの各要素の操作を確認するだけでは検証出来ない．品質を完全に保証するためには，あらゆる状況下における処理結果を確認しなければならない．しかし，検証作業の期間が限られており，すべての状況で処理を試行することは現実的ではない．ある程度利用場面を想定して検証作業を行う必要があり，次の作業の実施を計画した．

【フェーズ2：ユースケースに基づく検証】検証対象ソフトウェアを利用して行う業務のユースケースを作成する．そして作成したユースケースから，どのような状況下でどのような検証を行う必要があるかを明らかにし，ユースケースの一連の流れを明記したチェックリストを作成した上で，検証を行う．

このフェーズでは，まず製品導入を決定した企画部門が作成した製品開発ロードマップを参照し，検証対象ソフトウェアを発注元の製品ラインナップに加える目的を整理した．そしてその目的を実現するためのユースケースを導出した．なお，発注元の製品ラインナップに含まれることになるので，発注先が想定していない他の製品と一緒に利用するようなユースケースも導出した．そして，そのユースケースを満たす一連の操作手順に基づいて，チェックリストを作成した．

検証作業を実施するうちに，実行時の状況を変化させると，期待通りに実施出来ないユースケースが存在することが判明した．実際に検証したソフトウェアとは異なるが，例えば文書作成のための海外製のエディタを導入する目的で検証作業を実施していて，「ファイルを編集する」というユースケースが存在したとする．そこでこのユースケースに基づいて検証したところ，日本語文字列を含むファイルを編集出来るが，ファイル名に日本語文字列を含む場合にはファイルのオープンすら出来ない，という状況である※4．そこで，少なくとも基本的なユースケースについては，発注元で想定出来る範囲内で実行時の状況を変化させたときに，期待通りに実施出来ることを確認する必要が生じた．このため，フェーズ2と並行し次の作業を実施することを計画した．

【フェーズ3：基本ユースケースに関する重点検証】フェーズ2で作成したユースケースから，検証対象ソフトウェアを導入する際に最低限実行出来る必要がある基本的なものを抽出する．そして，導入対象ソフトウェアを利用する顧客に対して調査を実施する※5．その結果，想定出来る範囲で検証対象ソフトウェアが動作するいろいろな状況を用意し，抽出したユースケースを期待通り実施出来るかどうかを確認する．

なお，フェーズ1，フェーズ2のように，チェックリストを作成することで検証すべき項目を明確化出来るものに関しては，チェックリストを作成する担当者と，その作業を実施していく担当者に分け，チェックリスト作成作業に起因する思い込みがなるべく生じないように配慮した．

4 検証作業実施経過

4.1 （フェーズ1）UIの要素に基づく検証

3.1節で述べたように，検証対象のソフトウェアは規模が大きいため，ドキュメントに記載されているすべてのモジュールの動作を精査するのは困難であった．そこで最初に，自社製品への導入目的として利用されるモジュールを選別した．そのようなモジュールとして，全31モジュール中，13モジュールを選別した．

作業開始から6週目にかけては検証作業に取り掛かれるメンバが少なく，メンバのうちの2名によりユーザ向けドキュメントを参照しながらこの選別作業を実施した．ただし，どちらのメンバも他の作業と兼任であったため，本格的に検証作業を開始出来たのは，残りのメンバが加わった7週目以降であった．

フェーズ1では，選別したモジュールのUIの要素を検証した．この検証作業は，作業開始7週目から14週目にかけて実施した．

まずドキュメントを参照して，各UIの要素について，チェックリストを作成した．その際に，説明に間違いがあって，UIの要素の操作手順と一致しない場合には，ドキュメントの不良として不具合指摘を行った．また，ドキュメントの記載から漏れているUIの要素がいくつか存在した．その場合は，他のUIの要素と共通の部品であると考えることが出来，発注元で動作を想像出来るものについては，共通部品としてUIの要素のチェックリストを参照してチェックリストを作成した．発注元で動作を想像出来ないものについては，不具合報告のタイミングで発注先に質問を行い，回答後に追加でチェックリストを作成して検証作業を実施した．

ドキュメントの記述の例を図1に示す．これは，プロパティダイアログボックスを表示するための操作と，ダイアログボックス内に表示されたUIの各要素についての操作方法と，操作の結果として得られる出力を記述している．検証対象のソフトウェアは画面表示をしてユーザに操作させる手順が多く，たくさんの画面から構成されていたが，すべての画面表示について，ドキュメントには，このような操作方法に関する記述がされていた．

このドキュメントに基づいて作成したチェックリストの例を表3に示す．ドキュメントに記述されている操作を実施した際に，記載された通りの出力が得られるかどうかを確認出来るように，チェックリストを作成した．検証対象ソフトウェアには，ユーザ向けのドキュメントのみが添付

脚注

※4 この例で発注先は，日本への納入に向けてファイル編集のユースケースについて日本語での記述を考慮していたが，そのユースケースを実施する際の状況としてファイル名に日本語が含まれることを考慮していなかった．このことが，発注元の想定と異なっている．

※5 検証対象ソフトウェアを発注元の既存販売製品のラインナップに加えて販売する予定だったため，既存販売製品の顧客に検証対象ソフトウェアを販売することを想定していた．このため，販売パートナーから具体的な顧客環境をヒアリングした．

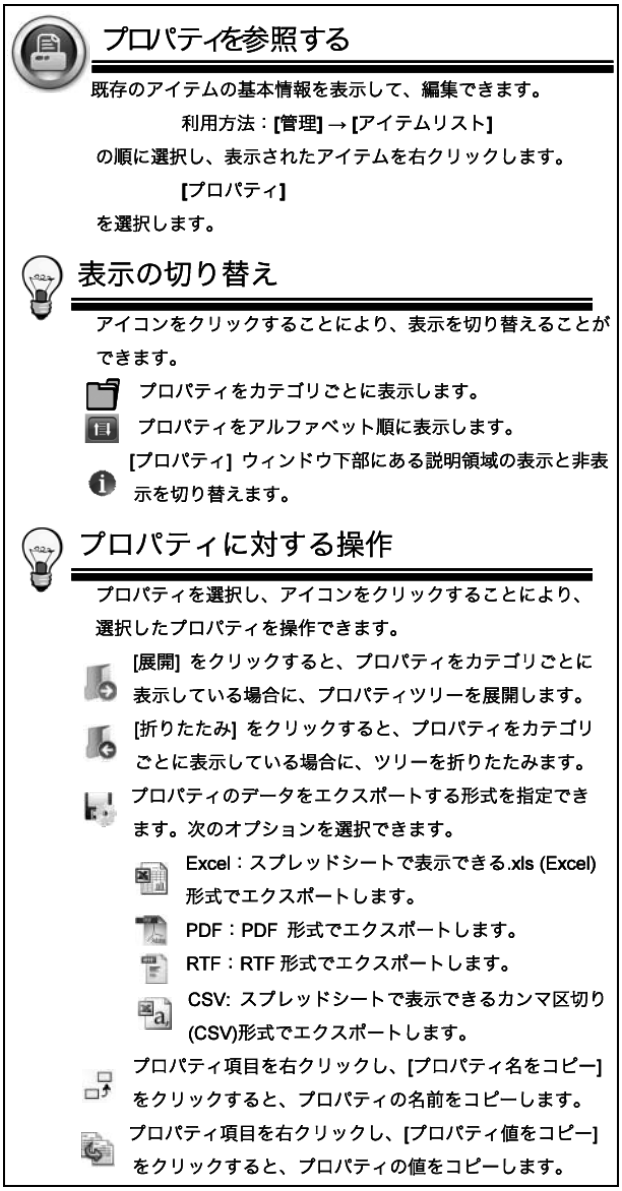


図1　UIに関するドキュメントの記述の例（抜粋）

表3　作成したチェックリストの例

#	関連する UI	チェックリスト内容のサマリ（以下の動作を確認）
1		ボタンをクリックすると，カテゴリ別のプロパティが表示される
2		ボタンをクリックすると，アルファベット順のプロパティが表示される
3		ボタンをクリックすると，プロパティウィンドウの下方の詳細表示エリアが表示または隠される
4		（カテゴリ別表示の状態で）ボタンをクリックすると，プロパティツリーが展開される
5		（カテゴリ別表示の状態で）ボタンをクリックすると，プロパティツリーが折りたたまれる
6		「エクスポート」を実行すると履歴がエクスポートされる（選択に応じて，Excel，PDF，RTF，CSV，XML の各形式で出力される）
7		プロパティの項目を右クリックし，「プロパティ名をコピー」によって，プロパティ名がコピー出来る
8		プロパティの項目を右クリックし，「プロパティ値をコピー」によって，プロパティ値がコピー出来る

表4　不具合報告件数
（カッコ内は各フェーズの報告件数合計に対する割合）

重要度	A	B	C	D	合計
フェーズ 1	0 (0%)	28 (18%)	76 (49%)	50 (33%)	154
フェーズ 2	0 (0%)	10 (10%)	54 (53%)	37 (37%)	101
フェーズ 3	2 (3%)	27 (44%)	31 (50%)	2 (3%)	62

されており，ソフトウェアの動作を理解するためにはそれを参照するしか手段が無かった．このためフェーズ 1 では，UI 操作を通して検証作業メンバがソフトウェアの動作を把握することも目的とした．

このような作業を実施し，755 件のチェックリストを作成した．そして，作業開始 9 週目から順次チェックリストに沿って検証を行った．なお後述の各フェーズも含め，報告者は不具合報告の際にその問題の重要度を，高い順に「A（導入の目的とする業務に致命的な影響を与えるもの）」，「B（致命的ではないが重大な影響を与えるもの）」，「C（与える影響は軽微だが修正する必要があるもの）」，「D（運用でカバー出来る場合など，修正する優先順位が低いもの）」の 4 段階で評価した．

表 4 に後述のフェーズも含めたフェーズごとの不具合報告件数を示す．フェーズ 1 ではドキュメントの不良に起因するドキュメントの記載を修正する必要がある不具合が多く発生したため，重要度 D よりも重要度 C の不具合が多くなっている．またフェーズ 1 では，UI に関するドキュメントのみを基に作業を実施した．複数の別画面で UI に対する別手順により実施される処理が同一データを利用する場合には，それらの処理が同じデータを基に実行されていることを検証結果から推測して把握する必要があり，状況の把握が困難であった．

4.2 （フェーズ 2）ユースケースに基づく検証

フェーズ 2 では，販売先として想定しているパートナー企業からの情報に基づき，具体的にこのソフトウェアを利用する部署を想定した上で，検証対象ソフトウェアを利用した業務に関するユースケースを作成した．そしてその際の操作に合わせて，ユースケースを満たす操作手順を，チェックリストとして挙げた．

例えば，「エディタによってファイルを編集する」という

ユースケースを作成したとする．このユースケースを満たすために必要な操作としては，単純に文字を入力したり挿入したりするような操作であったり，いろいろな文字列をコピーして貼りつける操作であったり，いくつもの操作が考えられる．それらの操作として考えられるものをすべて列挙し，チェックリストを作成した．

フェーズ 2 の作業は，作業開始 12 週目から 22 週目にかけて行われた．まず 13 個のユースケースを作成し，441 件の操作手順をチェックリストとして挙げた．そして，作業開始 14 週目から順次チェックリストに沿って検証を行った．

3.4 節で述べたように，フェーズ 1 ではある程度モジュールを選別してから検証を行った．フェーズ 2 で検証対象としたモジュールの一部には，フェーズ 1 で選別していないものが存在した．そのため，フェーズ 1 で検証を行わなかった UI を検証することがあり，フェーズ 1 と同様にモジュールの動作不良の不具合報告よりも，UI についてのドキュメント記述ミスに起因する不良という不具合報告が多くなった．このため，表 4 に示したように，不具合報告件数の重要度の割合はフェーズ 1 と同じような傾向になり，重要度 A や重要度 B の不具合の割合が小さくなっている．なお，検証対象ソフトウェアには UI に関するドキュメントしか存在せず，発注元の目的とするユースケースを実現するためにそのドキュメントを利用出来るかどうかは，UI の説明から推測する必要があった．

4.3 （フェーズ 3）基本ユースケースに関する重点検証

フェーズ 2 で洗い出したユースケースのうち，検証対象ソフトウェアを利用した業務の実施に最低限必要となるものを抽出した．その上で，発注元企業が想定した状況下で，抽出した個々のユースケースに含まれる操作を実施出来るか，フェーズ 2 と並行し検証した．

フェーズ 3 の作業は，作業開始 19 週目から 22 週目にかけて行われた．フェーズ 3 ではチェックリストを作成せず，検証対象ソフトウェアの基本的な処理が正常動作する状況を準備しながら動作確認を行った．

表 4 を参照すると，フェーズ 1 やフェーズ 2 に比べ，重要度が「B」,「C」の報告の割合が大きく，かつ「D」の報告の割合がとても小さくなっている．フェーズ 3 は，自社製品への導入に至る主目的を満たす処理の動作検証であり，不具合の存在が製品化に重大な影響を与えていることが分かる．フェーズ 3 では，発注元で満たすべきと考える品質に基づいて，検証工程末期にも関わらず単体テストの一部を実施したことになるため，発見された不具合の影響が比較的大きくなった．

4.4 検証プロセス全体での不具合報告の推移

フェーズ 1 及びフェーズ 2 では，最も多い不具合報告が，重要度「C」のもの，そして次が重要度「D」のものになっており，この 2 つで全報告の多くの割合を占めている．その一方で，フェーズ 3 のみが他と異なり，重要度「B」のものと「C」のものが同程度存在しており，この 2 つだけで全報告のほとんどを占めている．

検証対象のソフトウェアは，既に販売されている製品であり，発注先で一旦は検証しているはずである．フェーズ 1 及びフェーズ 2 では，発注先において動作すると想定していたことが見込まれる状況で検証を実施した．そのため報告された不具合は，日本導入にあたり作成されたドキュメントの内容の誤記載に起因するものが多く，フェーズ 1 とフェーズ 2 の全 255 件中 110 件がドキュメント内容の誤記載であった．

フェーズ 2 で作成したユースケースを実施する際に，発注先で想定されていない状況があることが推測出来たために，フェーズ 3 の検証作業が必要になった．フェーズ 3 の検証作業は基本的なユースケースに関するものであるにもかかわらず，フェーズ 2 の作業を実施するまでは作業の実施自体が計画されていなかったものである．このことにより，結果的に検証作業工程の後半になって重要度の高い不具合の報告が発生した．

全フェーズでの不具合報告状況推移として，期間全体を一週間ごとに区切った不具合報告件数の累積を図 2（a）に，フェーズごとの不具合報告件数の推移を図 2(b)に図示する．表 2 に示したように発注先への 3 回の指摘では，出来るだけ早い段階で重要な不具合を指摘出来，検証期間の後半では，不具合の発生状況が収束すると予測していた．しかし実際には，検証作業の遅い時期に重要度の高い不具合が多数発見された．とくに図 2（b）に示したように，フェーズ 3 については検証作業の終盤の短期間に実施したにもかかわらず，重要度 B の不具合が他フェーズよりも多く発見された．なおフェーズ 3 の検証作業は，当初の作業計画に対する追加の作業であったが，発見された不具合は修正しないと製品として出荷出来ない状態となり，結果として検証作業期間を約 1 カ月延長することになった．

5 OEM 製品検証作業プロセスの課題

5.1 対象にした検証プロジェクト事例の課題

2 章で述べたように，資本関係の無い企業間で Outsourcing を実施する際には，発注元と発注先の品質基準が異なるため，発注元において検証作業を実施することになる．本論文で例示した検証プロジェクトも，同様に発注元で実施した検証作業である．この作業の各フェーズにおいて，次の問題が発生した．

【フェーズ 1】 UI に関するドキュメントしか存在しないため，複数の別画面に対する手順により実施される別個の処理が同一データを利用する場合に，そのことを把握出来なかった．このため，ある UI について実施した検証と別の UI について実施した検証が同一データを利用して同一目的の処理を実施している場合に，状況の把握が困難になった．

【フェーズ 2】 UI に関するドキュメントしか存在せず，ユースケースを実現するためにどの UI を利用すべきかを UI の説明から推測する必要があった．

【フェーズ 3】 発注元で満たすべきと考える品質を発注先が満たさなかったため，単体テストの一部を実施する必要が生じた．

これらは，2 章で述べた次の状況に起因するものである．

状況 1）発注元が要求や設計に関わる情報を参照出来ない［VIDGER1996］.

状況 2）発注先において発注元と同程度の品質を満たすような要求に基づいて設計されていない場合が多い［BARBOSA2007］.

OEM によるソフトウェア供給関係では，発注によって開発が開始される通常の Outsourcing とは異なり，発注元の要求に基づいてソフトウェアが実装されるものではない．通常の Outsourcing では，自社の要求の確定後にソフトウェア開発が実施されるため，発注元の要求を満たしたソフトウェアが納入される．しかし OEM では，発注元の要求と納入されるソフトウェアの満たす要求が完全に一致しない場合がある．今回の事例では，発注元はある特定の新技術を提供するために OEM によって開発されたソフトウェアを自社製品に導入して販売したが，当該技術を利用するために必要な動作環境に対する想定が発注先と異なっていた．

更に OEM によって他社製品を導入する際には状況 1）に挙げたように，要求に関わる情報を発注元が参照出来ず，発注先がどのようなユースケースを想定しているのかを確認出来ない．今回の検証作業の初期段階では，発注元のユースケースにおいて想定する業務の実施に最低限必要になる基本的な処理が，発注先が想定するユースケースを実現する上でも必要な処理であり，十分に検証が行われているものと推測していた．しかし検証作業を進めていくにつれて，それらの処理について発注元の想定する状況に合致する検証が十分に実施されていないことが判明した．その結果，発注元のユースケースを実現するための基本的な処理に対する検証作業が作業工程の終了近く（フェーズ 3）に実施されて工程末期に重要な不具合が発見されることになった．発注元から発注先のユースケースを参照出来れば，発注元が想定するユースケースにおいて実行される基本的な処理を発注先のユースケースでどのように扱っているかを知ることが出来る．これにより，発注先が当該処理について，どのような想定で検証作業を実施したのかを正しく推測出来る．

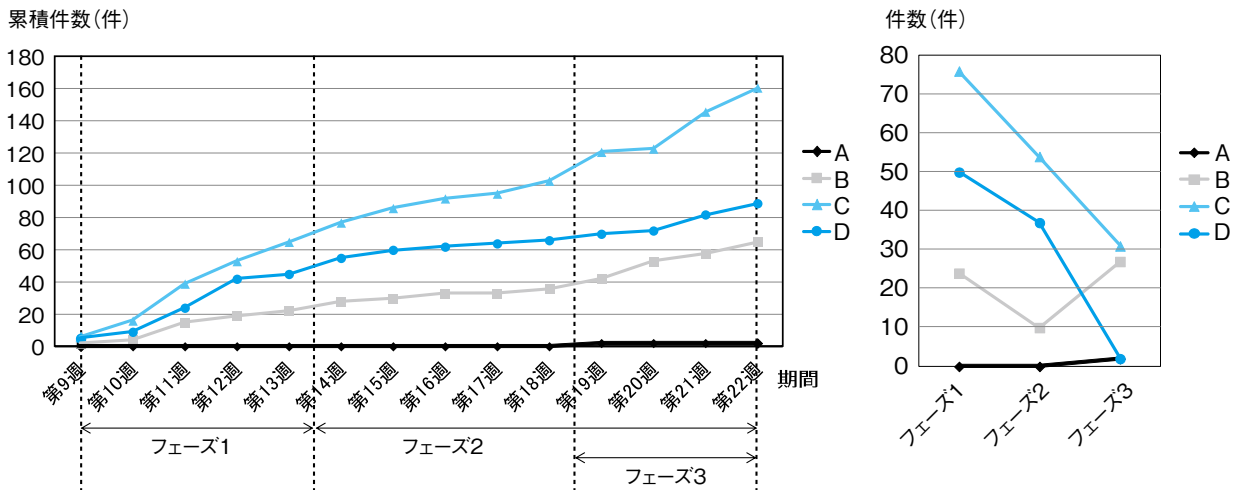
更に，状況 2）に記載したように，海外製ソフトウェアを導入する場合に，発注元と発注先でテストの品質基準が大きく異なる場合が多い．前述のエディタでのファイル編集の例では，日本への納入に向け日本語ファイル名の指定を想定して検証を実施することは発注元では当たり前の品質基準であったが，発注先ではそれが満たされていなかった．いくつかのモジュールの動作を確かめたところ，その当たり前の部分で品質が満たされていないことが判明し，結果的にフェーズ 3 で単体テストを追加的に実施した．このように今回の事例では，発注元の品質基準を発注先が満たしていなかった．すべての事例においてこのような状況にはならないが，発注元と発注先での品質基準の相違を，発注元で確認する必要がある．その際に，発注先でどのような基準に基づいて検証が実施されたかをあらかじめ知ることが出来れば，検証プロジェクトの早い段階でフェーズ 3 の単体テスト実施の要否を決定出来る．その結果，早い段階で重要な不具合を検出出来るようになる．

このように，OEM によって開発されたソフトウェアを自社製品へ導入する際の検証作業では，次の情報が発注先から発注元に開示されていれば円滑に作業を遂行出来る．

A） 要求定義や，想定したユースケースに関する詳細な情報（状況 1 に対応）

B） 単体テストの検証項目リスト（状況 2 に対応）

A）に関しては，通常は統合テストでの検証項目とほぼ一致するはずである．従って，単体テスト，統合テストに関する検証項目のリストと，それらが不十分な場合に備え追加的に要求定義に関する情報が開示されていれば，検証作業を円滑に実施出来たと考える．今回の検証作業ではフェーズ 3 の実施により約 1 カ月の作業遅延が発生した．フェーズ 3 では前述のように単体テストの一部を実施したと考えることが出来，他のフェーズとは独立して作業を実施可能であると考えられる．従って，早い段階でこの作業を実施出来れば，約 1 カ月の作業遅延の期間を抑えることが出来る．すなわち A）や B）の情報が開示されれば，フェーズ 3 で実施した検証作業追加の判断とそれに伴うスケジュール調



(a) 全期間での累積件数推移　　(b) フェーズごとの件数推移

図2　不具合報告件数推移

整を初期段階で実施出来，作業期間を１カ月近くは短縮出来たと考えることが出来る．

5.2　OEM ソフトウェア検証作業プロセス推進の指針

OEM によって開発されたソフトウェアを自社製品に導入する場合，今回事例として紹介した検証作業プロセスのようにプロセスの後半で重要度の高い不具合が発見されることを防ぐためには，前述情報を発注先から入手出来るように契約を行うべきである．それによって，次のような順序で発注先の想定していないユースケースから順に検証を実施することが出来る．

手順①　発注元のユースケースのうち，発注先が想定するユースケースと異なるものを抽出する．

手順②　手順①で抽出したユースケースを満たすために必要なモジュールの単体テストを実施する．

手順③　手順①で抽出しなかったユースケースを満たすために必要なモジュールの単体テストを実施する．

手順④　発注元の全ユースケースについて，結合テストを実施する．

手順⑤　発注先の想定するユースケースのうち，手順④で結合テストを実施していないものを満たすために必要なモジュールの単体テストを実施する．

手順⑥　発注先の想定するユースケースのうち，手順④で結合テストを実施していないユースケースについて，結合テストを実施する．

このような手順に基づいて検証プロセスを実施することにより，手順②において，発注元が想定しているユースケースのうち，発注先で想定していなかったユースケースに基づいて，モジュールの単体テストを実施出来る．今回対象にした検証プロジェクトでは，発注元が想定しているユースケースのうち，発注先で想定していなかったユースケースに関連する不具合の方が，結果的に重大な不具合になっており，これが検証プロセスの後半（フェーズ 3）で発見された．発注先で想定しているユースケースに基づいた単体テストは発注先で実施していると期待出来るため，手順②では発注先で想定していないユースケースに基づく単体テストを優先的に実施する．これにより，検証プロセスの初期段階で，発注先の想定していなかったユースケースに基づく重要な不具合を発見することが可能になる．

この手順を実施するためには，発注先から前述の A），B）の情報を入手する必要がある．パッケージソフトウェアなどの一般的に販売されるソフトウェアではなく，顧客の要求に合わせて開発されるソフトウェアでは，要求定義などに顧客の機密情報を含む場合があり，OEM 開発で供給される側の発注元では，これらの情報を入手出来ない可能性が高い．ただし，ある会社において顧客の要求に合わせて開発されたソフトウェアには顧客ノウハウが混入することが多いため，一般的にそのようなソフトウェアを別の会社が OEM で導入することは考えにくい．

仮にそのような状況がある場合でも，受発注者間の開発状況を可視化するための提案［MATSUMOTO2009］や，それを実現するための各種支援ツール群［INOUE2009］を利用することで，発注先がどのような想定で開発を実施したか，どのようなテストを実施していたかを，発注元で知ることが出来る．これらの提案は，「いつ，どこで，誰が，どのように」開発したものであるかというトレーサビリティ情報（ソフトウェアタグ）をソフトウェアに添付し，受発注者間で開発プロセスを確認する際の手段として利用出来るような社会基盤と，その実現を支援するツール群を提供している．Outsource 開発において，ソフトウェアタグにある情報を添付することが標準的になれば，要求に関する情報について匿名化した上で，受発注者間でやり取りされるようになる．従ってこのような基盤を整備すれば前述のような情報を発注先から入手することは実現可能であり，それによって発注元での導入作業を迅速に実施出来るようになる．

6　まとめ

本研究では，企業業務に利用される海外他社の OEM によって開発されたソフトウェアを自社製品に導入し，販売する際に実際に行われた検証プロジェクトの経過を観察した．その結果，導入の際の検証作業を効率的に推進するために，ソフトウェア製品とユーザ向けのドキュメントに加え，発注先で実施した単体テストや統合テストに関する検証項目のリストと，要求定義に関する追加的な情報の開示の効果を検討した．そして，それらを開示した場合の検証作業プロセスについて指針を述べた．

謝辞

本研究を推進するにあたって貴重なアドバイスをいただいた，日立製作所 飯塚大介氏，川田伸一氏，村上貴史氏に，深く謝意を表します．

参考文献

[BARBOSA2007] R. Barbosa, N. Silva, J. Duraes, H. Madeira : Verification and Validation of (Real Time) COTS Products using Fault Injection Techniques, 6th Int' lIEEE Conf. on Commercial-off- the- Shelf (COTS) - Based Software Systems (ICCBSS'07), pp.233-242, 2007

[ITO2010] B. Ito, E. Tomiura, R. Wakasugi : Does Firm Boundary Matter?: The Effect of Offshoring on Productivity of Japanese Firms, RIETID is cussion Paper Series 10-E-033, 2010

[INOUE2009] 井上克郎，楠本真二，飯田元：事故前提社会に向けたユーザ・ベンダ間での開発データ共有ーソフトウェアタグ規格とソフトウェアタグ支援ツールー，SEC journal, Vol.5, No.4, pp. 234-243, 2009

[MATSUMOTO2009] 松本健一：事故前提社会に向けたユーザ・ベンダ間での開発データ共有ー StagE プロジェクトとソフトウェアタグー，SEC journal, Vol. 5, No. 3, pp. 198-203, 2009

[MIC2007] 総務省情報通信政策局情報通信経済室：オフショアリングの進展とその影響に関する調査研究，総務省平成 19 年調査研究報告，2007.http://www.soumu.go.jp/johotsusintokei/linkdata/other017_200707_hokoku.pdf

[ROBINSON2004] M. Robinson, R. Kalakota : Offshore Outsourcing: Business Models,ROI and Best Practices, US: Mivar Press, 2004

[VIDGER1996] M. R. Vidger, M. W. Gentleman, J. Dean : COTS Software Integration: State of the Art, NRC-CNRC Report, National Research Council, Canada, 1996