B-009

テスト観点分析によるソフトウェア要求仕様の品質向上

Test View-Point Analysis for Development of Software Requirement Specifications

石井　俊直*1, 細谷　泰夫*1, 吉岡　克浩*1, 前田　融磁*1, 石原　鑑*1

Toshinao Ishii,　Yasuo Hosotani,　Katsuhiro Yoshioka,　Yuji Maeda,　Akira Ishihara

概要

ソフトウェア要求仕様の開発に於いてテスト観点分析を適用することで要求仕様の品質を向上する新しいテクニックについて提案する。このテクニックは、従来手法で特定した要求仕様に対して、要求仕様をテストするという観点の下、要求仕様が動作する際のシステムの外部を含めたコンテキストについての検討を行い、特定したコンテキストの中に要求仕様を投入する思考実験を行うことでテストケースを導くための体系立てた方法である。実際の開発プロジェクトに本手法を適用したところ、ユースケースのシナリオステップ数とテスト観点分析によって導かれたテストケース数に高い相関はなく、本手法によってユースケース分析では抽出できなかった要求仕様がテスト観点分析によって特定されることが観察された。

1.　はじめに

ソフトウェア開発プロセスにおけるソフトウェア要件定義は、共通フレーム 2007 [1] によると、上流から与えられる要件を設計が可能な技術的要件に変換することが目的であると定義されている。従って、技術的な詳細度においてはこのフェーズは最上流に位置し、プログラムのテストを行うようにソフトウェア要件をテストしてその妥当性を確認することは容易ではない。古典的なＶ型のソフトウェアライフサイクルモデルでは、上流になるほど要求や設計とテストによる確認との間に多くの活動を含むこととなり、要件定義のテストによる確認にはそれだけの手間がかかってしまうことになる。これに伴いテスト結果が否定的である場合の影響範囲が広がってしまうことも大きなリスクである。

このように、定義するソフトウェア要件の品質は非常に重要な要素であり、その向上のために様々な取り組みが行われている。この取り組みを、(1) プロセスのレベル、すなわち、作成する成果物の定義・成果物間の関係定義・成果物を作成する活動やステイクホルダの定義とのレベル、(2) テクニックのレベル、すなわち、個々の成果物を作成するための有効な方法やツールのレベル、(3) 開発者の能力のレベルに分けて考えることができる。(1) の例としては、先に述べた共通フレーム 2007 などプロセスの体系化がある。RUP [2] も共通フレームとは異なるスタイルの開発プロセスモデルであり (1) に分類でき、extreme programing [3] などアジャイル開発を具現化したプロセスもこの例と言える。(2) の例は、開発対象ドメインに依存するなどの理由もあって種類も量も多く、ソフトウェア開発の多くの著作はこのレベルの取り組みと捉えることができる。(3) は OJT を含めた様々なトレーニングやスキルセットの標準化などがその例である。

本論は、主に (2) テクニックのレベルに関するものである。(2) についてソフトウェアの要件定義に焦点を絞り、僅かだが代表的取り組み例を挙げると、共通フレームを元にした [4] では、ソフトウェア要件定義として作成すべき成果物をより詳細に定義するというアプローチをとっている。RUP に代表されるオブジェクト指向エンジニアリングでは、ユースケース分析やバウンダリ分析などの方法を通して品質の高い要件定義に取り組んでいる。アジャイルプロセスでは、extreme programing に代表されるように、要件定義からテストまでを短期間で行いテストのフィードバックを開発に積極的に取り入れるというスタイルがとられる。

本論で提案するテクニックは、テスト観点分析と呼ばれる最近テストの設計に用いられるようになった考え方を、ソフトウェア要件定義のフェーズで要件の品質向上に寄与するように応用したものである。このテクニックの適用には、開発プロセスの全体的スタイルを変更する必要はなく、例えばユースケース分析結果の品質向上といったように、限定的な範囲で適用できる。この手法の背景にある原則は、設計とテストの並行実施という考え方である。この考え方について Spillner [5]

*1: 三菱電機 (株)

は、V 型開発モデルを拡張した W 型開発ライフサイクルモデルにおいて「残念なことに V モデルでは明らかに表されていないが、要件や設計の仕様が定義された時点で、テストケースが定義されなくてはならない。」と一般的に述べているが、本論ではソフトウェア要件定義のフェーズに関してこれを実現する具体的方法を提案する。また、この手法を実際のソフトウェア開発プロジェクトに適用した事例についての報告を行い、提案手法についての議論を行う。

2. ソフトウェア要求仕様

本節では手法提案の準備として、ソフトウェア要求仕様についてのモデル化を行い、またソフトウェア要求仕様の具体的表現として用いるユースケースモデルについて簡単に定義する。

2.1 要求仕様のメタモデル

共通フレーム 2007 では、「ソフトウェア要件定義」という用語が用いられている。要件定義は要求が仕様化されたものと解釈できるが、本論では要求とこれを仕様化したものを区別するため以下で述べるモデルを採用し、要件定義という用語はこれから先では用いないこととする。

本論で採用する要求仕様のモデルは、Paranas らのモデル [6] を元にしている。図 1 に示すようにソフトウェア製品を利用者が実際に使ってる状況では、利用者には何らかの課題があってその課題を達成するためにソフトウェア製品を操作し、ソフトウェア製品が出力する情報を結果として受け取る。このとき、自然に、利用者とソフトウェア製品の間にはインターフェースが存在することになる。従ってソフトウェアの内部、インターフェース、ユーザの課題という三つの領域を特定することができる。本論で、要求とはユーザの課題を表す概念でありこれが含まれる空間を課題領域と呼ぶことにする。インターフェースを定めたものが要求仕様であると考え、ソフトウェアの内部については、ソフトウェアの設計や実装を含むソリューション領域と捉える。この、外部–インターフェース–内部というモデルはインターフェースの場所によって様々なレベルに適用できるメタレベルのモデルであるが、本論では、先に示した利用者課題–ソフトウェア要求仕様–ソフトウェア設計のレベルで固定する。なお、このインターフェースすなわち要求仕様はソフトウェア設計という技術的領域との境界と考えると、共通フレーム 2007 のソフトウェア要件定義の「設計が可能な技術的要件」という定義とおよそ一致するものである。

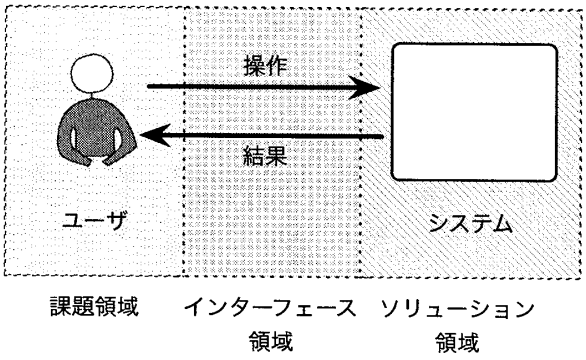


図 1 要求仕様のモデル。要求仕様は、ユーザとシステムの間に自然に生じるインターフェース領域を定義するものと考える。

3. ユースケースによる要求仕様定義

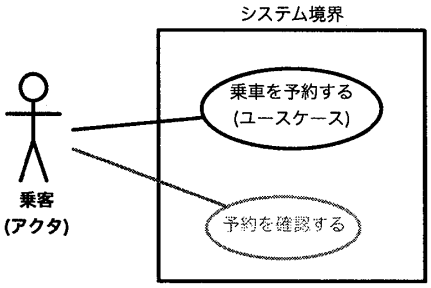
提案する手法を、ソフトウェア要求仕様の具体的な定義を使ってに提示するため、ソフトウェア要求仕様の定義にユースケースを利用する。要求開発にユースケースモデルを採用する利点は様々なことが知られているが、それら利点は本論においては本質的ではない。よく知られた課題領域を例として使うことでソフトウェアの内部にほとんど触れずに単純な実例を示すことができる説明の便宜上ユースケースモデルを使う。

ユースケースモデルの詳細は [7], [8] などが詳しいが、簡単に述べると、利用者 (アクタ) の課題が解決されるまでのアクタ (一般にはアクタは複数) とシステムの間で起こる振る舞いを表したものがユースケースである。ユースケースはアクタの観点で表され、記述のスタイルにもよるが、ユースケースを開始する主アクタの目的、ユースケースが開始される条件なども明確化される。通常、非形式的記述、すなわち自然言語、により表される。ソフトウェア要求仕様開発のような上流のフェーズでは、インターフェースがシステムの設計に依存しないため、図 2 のように具体的な振る舞いというよりは、アクタとシステムの意図のやりとりとなる。

ユースケースにおけるやりとりを定義した文書は、ユースケースシナリオと呼ばれるが、ユースケースシナリオにはデータを表す名詞や振る舞いを表す動詞、また、それらの順序が定義される。ユースケースをこのように捉えると、ソフトウェアの機能を表した機能仕様との間で変換できることがわかる。

4. 要求開発におけるテスト観点分析

本節では、テスト観点についての定義を行い、ソフトウェア要求仕様開発におけるテスト観点分析の適用するテクニックについて具体例も使い議論する。



ゴール	乗車を予約する
主アクタ	乗客
主シナリオ	1 乗客は、乗車システムにログインする。
	2 システムは、乗車希望条件入力画面を表示する。
	3 乗客は、希望条件を入力する。
	4 システムは、希望条件に合う運行予定を表示する。
	5 乗客は、表示された運行予定から乗車予約希望を選択する。
	6 システムは、乗車予約を行う。
	7 システムは、乗車予約情報を表示する。
拡張シナリオ	4a 乗車希望に見合う運行予定が無い場合…

図 2 ユースケースのサンプル。上図は、ユースケース図で、ユースケースをユースケース名 (アクタの目的) を囲んだ楕円で表す。下表はユースケースシナリオ。アクタとシステムのやりとりをシンプルで明快な自然言語で表す。

4.1 テスト観点

Spillner は要求仕様が定義された時点で、テストケースが定義されるべき (Test cases must be specified once the corresponding requirements and design specifications are available.) と言ったが、少なくともソフトウェア要求仕様のレベルであってもそれがテスト可能であるように定義されることが望ましい。更に、テストが可能であるように要求仕様を定義すると、テスト観点の分析を通して多面的に評価することで定義された要求仕様を揺さぶり、ある種のテストを行うことができる。

テスト観点という概念は比較的新しく、標準化はまだ行われていない。西 [9] は、「テスト対象のもつテストすべき側面、テスト対象が達成すべき性質、テスト対象をテストの立場からモデリングしたもの」と説明し、この概念をテスト設計プロセスに応用している。吉澤 [10] は西の定義を引用しさらに分析を加えて、「テスト観点はテスト対象と前提条件に分かれる。」として具体的なテスト設計を進めた事例を報告してる。西の定義にある「側面」や「性質」には、テスト設計では当たり前であるところの「多様な」あるいは「あらゆる」という性質が込められているように我々は考えている。実際に吉澤は報告の中で、テスト観点の「列挙・整理・検討・詳細化・体系化」について述べている。

テスト設計ではなく、ソフトウェア要求仕様開発にテスト観点分析を応用するための定義として、我々は「ソフトウェアの実行時のコンテキスト」をテスト観点の母集団と考えることにする。この「コンテキスト」は西らのテスト観点の定義よりも広い範囲を指している。「テストすべき」という限定を外したのは、要求仕様開発者向けの配慮でもあるが、以下で提案するテクニックでは、ソフトウェア実行時のコンテキストを、はじめテストすべきか否かで限定せずに洗い出し、それに以下で具体的に述べるテスト観点分析操作を加えることでソフトウェア要求仕様の充実化を行うためでもある。コンテキストという主に外部を示す用語を採用したのは、ソフトウェア要求仕様開発では概念上はソリューション領域に立ち入らないためである。概念上、ソリューション領域、すなわちソフトウェアの内部について定義が行われないので、その詳細に関わるようなことは分析ができない。なおコンテキストとは要求仕様、すなわちインターフェース、に対するコンテキストであるので、実際の開発でソフトウェア要求仕様がソフトウェア内部に関する内容を含んでもそれは概念上はインターフェースの定義が変わるだけであって、このテスト観点の定義が矛盾することはない。

4.2 具体例

テスト観点分析の議論で利用する例を示す。開発対象例として以下の単純化した電話機のソフトウェアを用いる。まず電話機の構成は以下のものとする。

- 電話機は、入力パネル、表示パネル、鳴動装置を有している。
- 電話機は、ネットワークポートを有している。
- 電話機には、マイクロプロセッサが搭載され、その上で電話機制御ソフトウェアが稼働する。
- 電話機制御ソフトウェアは、入力パネルドライバ、表示パネルドライバ、鳴動装置ドライバと接続している。
- 電話機制御ソフトウェアは、ネットワークドライバと接続している。

次にソフトウェア要求仕様の一部「電話をかける」をユースケースで表すと以下のようになる。なお、ここではドライバを電話機制御の外部のアクタと捉えている。

ユースケース名
　電話をかける
主アクタ
　入力パネル (ユーザ)
その他のアクタ
　ネットワークドライバ

基本系列

1. 入力パネルは電話番号を入力する。
2. 電話機ソフトウェアは、鳴動装置で入力番号相当の音声を鳴らす。
3. 電話機ソフトウェアは、表示パネルに入力された電話番号を表示する。
4. 入力パネルは、通話開始を入力する。
5. 電話機ソフトウェアは、ネットワークと接続を開始する。

4.3 ソフトウェア要求開発へのテスト観点分析の応用

オブジェクト指向エンジニアリングでは、ユースケース分析結果を入力としてバウンダリ分析を行い要求の形式化を進めることがよく行われる。このような順方向の作業、すなわちプログラムという機械が実行できる完全に論理的な成果物に向いた作業では、形式化によって入力成果物の論理的矛盾を検出することは難しくないが、入力に抜けがあった場合にこれを検出することは容易ではない。なぜなら、作業では入力成果物を体系全体と捉えるからである。従って、表現形式が何であれ要求仕様を定義する場合には、下流工程に作業を渡す前に抜けが無いかどうか注意深く考慮する必要がある。

成果物をレビューするというテクニックが抜けを防止する目的で広く利用されているが、このテクニックは個人の能力に依存する面が大きい。本論ではより系統立ったテスト観点分析の応用について提案する。なお、要求仕様がユースケースとして表されていると想定して説明を行うが、テスト観点分析の対象はユースケースに限られるものではない。

このテクニックでは、まずはじめにユースケースが実行された場合にシステムの動作に関連するコンテキストの抽出を行う。このコンテキストは、大別すると、システムの実行に関連するパラメータとシステムの実行に影響する条件あるいはその結果である。ユースケースシナリオにはパラメータの候補となる概念が表現されているし、その条件文や動詞としてテスト観点としての条件や結果が表現されている。しかしながら、これに限ってテスト観点分析を行うのではテスト観点分析で得られるものはあまりない。テスト設計者が網羅的なテスト設計を目指すのと同様に、ユースケースシナリオはシステムの実行空間の一部分でしかないと捉えてより広く関連するコンテキストの抽出を行う。

電話機のユースケースに関して、テスト観点の例を列挙すると、テスト観点となり得るパラメータには、番号キー、キー入力待ち時間、音量設定、バッテリー残量、時刻などが考えられる。また条件としては、入力番号、トーン鳴動音、番号表示、接続制御、接続結果などがある。時刻を例にとると、これをテスト観点の候補として抽出したのは、時刻によって番号表示における明るさや鳴動における音量といったものが影響を受けるかもしれないからである。

表１　「電話機」ユースケースに関するテスト観点分析のパラメータ組合せの例。「↑」は上に同じを意味する。「*」は任意の値を意味する。

ID	音量設定	時刻	バッテリー	待ち時間
A	*	昼/夜	閾値以上	閾値以下
B	↑	↑	↑	閾値以下
C	↑	↑	閾値以下	閾値以下
D	↑	↑	↑	閾値以下

これらの観点は、パラメータであれば水準によって細分化することができ、条件/結果であればその内容によって細分化することができる。例えば、パラメータとしての時刻について昼間と夜間を、バッテリー残量については閾値以上と閾値以下を、条件としての入力番号について、番号/開始/終了/クリアを、結果としての接続制御には現状位置/開始/終了といった細分化ができる。

このようにユースケースシナリオにとらわれず広くコンテキストを抽出することは、要求を揺さぶることに相当する。言い換えるとこれは、論理的な整合性よりも、システムが稼働するであろう環境をなるべく現実的に想定して、その環境にユースケースを投下してみるといった思考実験的な作業であり、分析的な作業と対比される発散的な性質を有している。

次に、抽出したテスト観点を組み合わせる。組合せを作る目的は、テストケースを導くことである。ここでは、一方でパラメータの網羅的組合せを作成し、他方で条件/結果から処理フローを定義し、これらを付き合わせることによってテストケースを導き出すテクニックの一例について述べる。一般的には、同様の目的のために、テストケースを定義する方法は様々なものがあり得る。

異なるパラメータの特定された水準を組み合わせることで、パラメータ組合せを作成できる。組合せ方は、全組合せを作っても良いし、場合数が多いときにはオールペア法や直交法などによって組合せ数を制限しても良いだろう。パラメータ組合せ例を表１に示す。

パラメータ組合せとは別に、条件/結果をもとに動作フローを特定する。動作フローはパラメータの場合と異なり、個々の条件や結果同士の依存性が強いので単純な組合せでは意味のある動作フローを効率的に特定することが難しい。このような課題には CFD [11] などの手法が知られているのでそうした手法を用いるべき

表 2 「電話機」ユースケースに関するテスト観点分析の動作フローの例。「–」は当該フローにその項目が無関係であることを示す。

ID	入力番号	トーン鳴動音	番号表示	接続制御	接続結果
1	番号	番号音	入力番号	維持	–
2	開始	開始音	開始	開始	–
3	終了	終了音	終了	終了	接続中
4	終了	–	クリア	–	接続中
5	クリア	–	接続状態	–	–
6	–	–	接続状態	–	接続中
7	–	–	接続状態	–	成功
8	–	失敗音	接続状態	–	失敗

表 3 「電話機」ユースケースに関するテスト観点分析のパラメータ組合せと動作フロー突き合わせの例 (一部)。「○」はパラメータの範囲で動作フローが起こりえることを意味する。

		動作フロー							
		1	2	3	4	5	6	7	8
パラメータ	A	○	○	○	○	○	○	○	○
	B						○	○	○
	C	○	○	○	○	○	○	○	○
	D						○	○	○

である。動作フローの例を表 2 に示す。

パラメータ組合せと動作フローを付き合わせることでテストケースを定義することができる。すなわち、パラメータ組合せのそれぞれは動作フローの一つあるいはそれ以上と結びついているか、あるいは意味のない組合せであるかのいずれかである。もしも意味のあるパラメータ組合せに結びつく動作フローがないのであれば、動作フローの分析が不充分であると考えられる。また、水準に差異に依らずに同じ動作フローに結びつくようなパラメータがあれば、そのパラメータはテスト観点としては考慮しなくても良いということになる。パラメータ組合せと動作フローの突き合わせ例を表 3 に示す。

パラメータ組合せと動作フローの突き合わせは、起こりえるケースを特定したものであるが、これを一つ一つ分析すると、抜けていた要求仕様が見えてくる。例を使って示すと、条件組合せ C,D ではバッテリー残量が閾値以下であるので、接続開始制御を行う 2 や接続中である 6 のフローでは、バッテリー切れに備えて接続開始を抑制するあるいは接続の終了といった処理を要求仕様に加える必要や、また、全ての動作フローにおいて時刻に応じて表示の明るさを調整する要求仕様が必要があるかもしれない。表 3 ではパラメータと動作フローの組合せのみを考えたが、この他にも動作フローの並行実行の組合せを考えることもでき、表 3 と同様に起こりえる組合せを特定し、それらケースについて分析を加えることで、例えば優先順位などの、追加すべき要求仕様が抽出され得ることは容易に想像できる。

4.4 テスト観点分析応用のまとめ

テスト観点分析は、ソフトウェア要求仕様の抜けを抑えるために有効なテクニックである。ソフトウェアが動作するリアルな環境を想定しそれをコンテキストとして明確化し、思考実験としてその環境に要求仕様を定義したソフトウェアを投入してみることでテスト観点を分析してテスト観点の意味のある組合せを導き、それらを付き合わせることでテストケースを作成する。次節で適用例について述べるが、一般的傾向について述べると、テスト観点分析の結果得られるテストケースは、ユースケースシナリオで定義されていないケースを含んでいる。それは、そもそもユースケースシナリオは、ケースを網羅的に抽出することを目的とせず、アクタのゴールを明確化することが目的とされているからである。ユースケース記述あるいはこれに相当する成果物について単独での網羅性を求めると、結果として複雑になるあまりモデル化本来の目的を達成しにくくなる。従って、テスト観点分析結果は、独立した成果物としてソフトウェア要求仕様を構成する要素と考えるべきであって、テスト観点分析結果と整合するように複雑化したユースケースを成果物とすべき理由は見あたらない。

5. テスト観点分析の適用例

本節では、要求仕様開発にテスト観点分析を実際に適用した開発プロジェクト事例について報告し、その結果についての考察を行う。

本例のプロジェクトで開発したシステムは、Web を用いた専用業務システムでユーザインターフェースを含む端末と、業務サーバ群からなるものであった。以下の報告はこの内の端末システムの要求仕様定義に関するものである。このシステムでは、数種類の専用業務を行うためのもので、中にはリアルタイム性が求められる業務が含まれている。要求仕様開発に従事した開発者は主に 4 名で、テスト観点分析に入るまでの要求仕様をおよそ一ヶ月の期間[1]でユースケース分析を中心にソフトウェア要求仕様を開発した。

まず開発したユースケースの規模について述べると、ユースケース図の数は 19 で、ユースケースあたりの主

1: ただしこの期間すべてで専念して本開発を行っていたわけではない。

表 4　適用例における成果物規模。列は左から ID、主シナリオステップ数、副シナリオ数、動作フロー数、テストケース数。L1~L4 は、テスト観点分析において動作に関する条件や結果を分析し、木構造のグラフの 形で表した際のノード数。L0(中心) をユースケースとし、Lx の x はユースケースまでの距離。

ID	steps	sub sc	flow	test	L1	L2	L3	L4
a	22	0	4	4	1	4	3	
b	14	5	8	12	5	9	13	
c	9	7	3	4	3	4	2	
d	4	2	3	4	3	5		
e	7	2	4	7	4	6		
f	15	8	6	29	4	9	14	17
g	4	1	4	6	3	6	2	
h	7	6	5	6	3	5	7	
i	10	5	6	9	3	8	14	
j	13	1	5	7	4	3	5	
k	12	8	4	4	3	5		
l	17	2	7	11	4	10	19	
m	3	1	4	4	2	3	3	
n	11	2	6	12	3	8	16	8
o	6	2	5	7	1	7	6	
p	3	0	3	3	2	3		
q	17	11	8	19	4	11	21	12
r	2	0	3	3	2	3		
s	15	0	6	17	6	13	13	6

シナリオステップ数は 2〜22 で平均は 10、副シナリオ数は 0〜11 ああった。これに関するテスト観点分析では、特定された動作フロー数は 2〜19 で平均が 7、テストケース数は 3〜29 で平均が 9 であった。システムの実行に影響する条件や結果はそれぞれを木構造のグラフの形で表したがその規模は表 4 に示すようなものであった。

このデータを元にテスト観点分析によって特定されたテストケースが依存している要因やその度合いについて知見を得るため、以下二種類の座標を図示したものが図 3 である。二種類は (1) (x,y) = (ユースケースのステップ数, テストケース数)、(2) (x,y) = (動作フロー数, テストケース数) である。この図から見られる傾向は、

- 特定されたテストケース数は、ユースケースシナリオのステップ数とはあまり相関がない。
- 特定されたテストケース数は、特定された動作フロー数と相関が強い。
- 従って、ユースケースシナリオのステップ数と動作フロー数は相関があまりない。

本来、ユースケースシナリオはアクタとシステムの間で起こるやりとりを定着したものであるから、ユースケースシナリオは動作フロー種類だけ存在しているべきである。しかしながら、ユースケースモデリングとはそもそも、動作を網羅的に特定することが目的ではなく、また、表現形式は網羅的な特定に向いているものでもない。従って、データに関するここで得られた傾向が見られたとしても、それをもってユースケース記述の品質の問題であると言うことはできない。具体的には示さないが、試しに今回開発したユースケースシナリオを動作フローを網羅するように書き換えると、Cockburn [7] が示す指標に照らすと複雑過ぎるユースケースシナリオができあがってしまった。それは本来のユースケースモデリングの目的から外れているというほかなかった。従って、本論で提案したテスト観点分析の主な目的はユースケース分析結果の問題点を明らかにするものではなく、ユースケース分析結果を補う詳細化と捉えるべきである。

6. おわりに

本論では、ソフトウェア要求仕様の開発に於いてテスト観点分析を適用することで要求仕様の品質を向上する新しいテクニックについて提案した。要求仕様を定義した段階でその妥当性を確認することは容易ではない。分析的なアプローチによって確認できることは、定義を体系全体として捉えるため、主に要求仕様の矛盾に関わることであり、要求仕様に対する分析によって要求の漏れを特定することは難しい。これに対して、本論で提案したテクニックは、要求仕様を従来法で特定した際の観点とは異なる要求仕様をテストするという観点の下、要求仕様が動作する際のシステムの外部を含めたコンテキストについての検討を行い、要求仕様を特定したコンテキストの中に投入する思考実験を行うことでテストケースを導くというものである。

このテクニックを実際の開発プロジェクトに適用した事例について報告した。この適用例を見る限り、ユースケースシナリオのステップ数、すなわちユースケースの複雑さと相関があると考えられる変数と、テスト観点分析の結果特定されテストケース数との相関はそれほど高くなく、テスト観点分析で特定された動作フローの原因や結果の場合数と特定されたテストケース数との相関が高かった。動作フロー数とテストケース数の相関が高いことは、前者を整理したものが後者であることを考えると明らかな事実かも知れない。むしろ重要なことは、ユースケースの複雑さと特定されるテストケース数の相関が低かったことであった。すなわちこの結果は、ユースケースの内容に対する依存性が低い事柄がテスト観点分析によって抽出されていることを示唆している。ユースケース分析がシステムの動作を網羅的に記述することを目的とする分析ではなく、また、実際に特定されたユースケース個々の内容を分析すると、ユースケースへのフィードバックが難しい要求仕様がテスト観点分析において特定されていた。

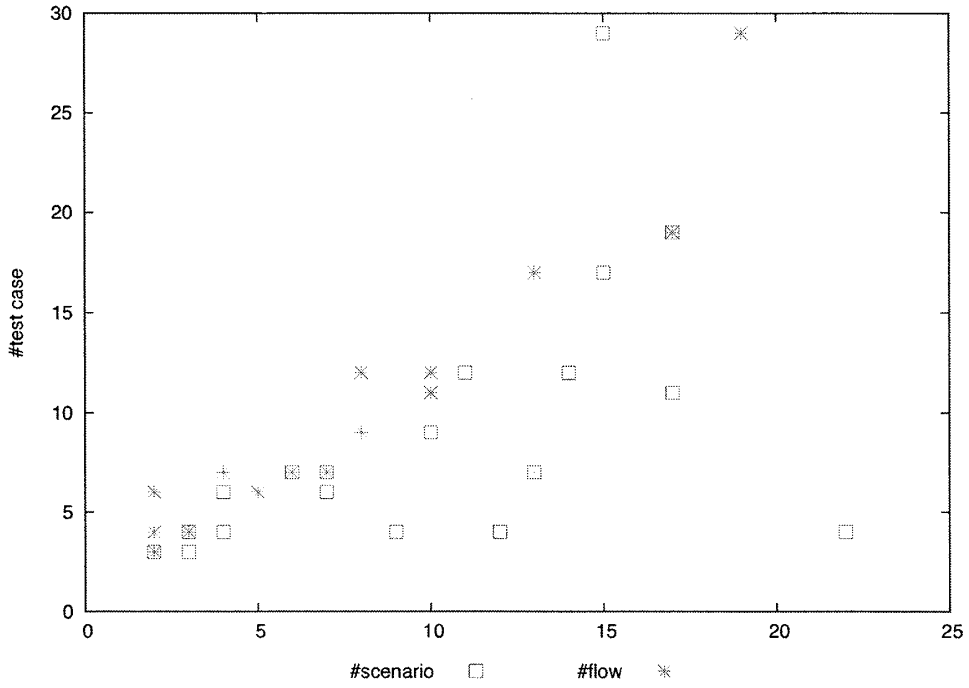


図 3　ユースケースシナリオステップ数と動作フロー数それぞれのテストケース数に対する相関。□は (x,y) = (ユースケースのステップ数, テストケース数)、＊は (x,y)=(動作フロー数, テストケース数) のプロット。

従って、この結果をユースケースの品質が低かったということはできない。テスト観点分析によって得られることは、ユースケース分析では特定が難しく、しかしながら要求仕様としては特定すべきことであると解釈できる。

今回提案したテクニックは、要求という曖昧な情報をユースケースという手法によって一度分析・整理して定着した成果物に対し、テスト観点という擾乱を加えるというものである。言い換えると、本論で提案したテクニックは、分析的に得られた成果物を、その分析時の想定よりも広い現実を作ってその中で何が起きるか実験するという発散的なプロセスであった。このように、一般論として、分析的な集約と実験的な発散とを組み合わせて実施することが、システム開発において有用であると考えられる。

参考文献

[1] ソフトウェア・エンジニアリングセンター編: 共通フレーム 2007, オーム社, 2007.
[2] ヤコブセン, ブーチ, ランボー: UML による統一ソフトウェア開発プロセス, 翔泳社, 2000.
[3] ベック: XP エクストリーム・プログラミング入門, ピアソン・エデュケーション, 2000.
[4] 日本ユニシス情報技術研究会編: システム開発の体系, 東京電機大学出版局, 1999.
[5] Spillner, A., et. al.: Software Testing Pactice- Test Management, Rocky Nook, 2007.
[6] Paranas, D.L. and Madey, J. : Functional documentation for computer systems. Science of Computer Programming, 25(1), p41-61, 1995
[7] Cockburn, A.:Writing Effective Use Cases, Addison-Wesley, 2001.
[8] Ambler, S.: Agile Modeling, Wiley, 2002
[9] 西康晴:テスト設計におけるモデリングのための記法の提案, JaSST'06 in Tokyo, 2006
[10] 吉澤圭介:テスト観点に着目したテスト設計 の実践事例, JaSST'07 in Kansai, 2007
[11] 松尾谷徹:CFD 方の極意, ソフトウェアテスト Press, vol. 8, pp. 98-107, 技術評論社, 2009