GSLetterNeo Vol.57
2013年4月

ユースケースモデリング（２）

オブジェクトモデリングスペシャリスト
土屋　正人
Masato Tsuchiya
m-tsuchi@sra.co.jp

UML が登場してから 15 年以上経ちますが、現在 13 種類ある UML の図（ダイアグラム）の中で最も利用されているのはユースケース図でしょう。ユースケース図と個々のユースケースの仕様（ユースケース記述）をセットにしてユースケースモデルと呼びます。いまさらユースケースモデルの説明など不要という人もいるかもしれませんが、初心に帰って、ユースケースモデルについて少し記してみたいと思います。

◆ユースケースモデルの目的

ユースケースモデリングでは、ユースケース図で外部から検証できるシステムの振る舞いを定義し、ユースケース記述でシステムとシステム外部とのやり取りを定義することで、システムの利用法（ユースケース）を明らかにします。システム利用者や連携する他システム、デバイス等、システムに関与する外部の存在（アクタ）を特定することで、システム化対象範囲が明確になります。

◆ユースケースの粒度

ユースケースの粒度については様々な見解があり、残念ながら統一されていません。これについては、GSLetterNeo Vol.7「ユースケースモデリング」をご参照ください。

◆アクタ候補の識別

アクタは開発対象システムの利用者・協力者であり、システムから見た外部の存在の役割を示すものです。

アクタは主として業務フローから識別します。業務フローから、システムにデータ／情報を与える、あるいは受け取る「人」「他システム」「機器」等を、アクタ候補として識別します。

◆ユースケース候補の識別

ユースケースは機能要求で、開始してからアクタにとって価値のある結果を返すまでがひとつのユースケースの単位となります。つまり単独で実行できる単位であり、他のユースケースと組み合わせて順番に実行しなければ意味を成さない場合は見直しが必要です。

ユースケースは主として業務フローに定義された、システムを利用するタスクから識別します。業務フローから、アクタからデータ／情報やイベントを入力あるいは出力するタスクを、ユースケース候補として識別します。同一アクタが関わるタスクが連続実行される場合、これらのタスク群はまとめてひとつのユースケース候補となります。

ユースケースモデリングに限ったことではありませんが、一度のモデリングで適切な単位（粒度）の識別が出来ることは稀で、通常、何回かのレビューを行って決定することになります。

◆ユースケース記述

ユースケース記述も様々なテンプレートがありますが、以下の項目を定義することで概ね一致しています。

- **ユースケース名**
- **概要**
- **事前条件**
- **事後条件**
- **イベントフロー（基本フロー、代替フロー）**
- **特記事項**

概要は、ユースケースを開始するアクタ、ユースケースの目的、タイミング（開始、終了）、内容などを1～3段

落程度にまとめます。

事前条件は、ユースケースを実行する前のシステムの状態を記述します。

事後条件は、ユースケースの実行が終了したときのシステムの状態を記述します。

事前条件はひとつ(1セット)ですが、事後条件はひとつ(1セット)以上必要になります。これはユースケースの終わり方が、正常終了の基本パターン、別パターン、異常終了など、複数のパターンが考えられるからです。

事前条件と事後条件に照合することによって、ユースケースで示された機能要求が満たされるかどうかを確認することが出来ます。すなわち事前条件と事後条件はテスト仕様書に利用することができます。これらは単にテストに使えるという利点に留まるものではありません。**事前条件と事後条件が定義されていない、あるいは定義できないということは、その要求の確認方法がないわけですから、要求定義として不十分**ということになります。

イベントフローは、ユースケース記述の中核となるもので、アクタとシステムとのやり取りを、曖昧さのない簡潔な文章で記述します。一般的に、まずアクタ側の記述から始め、次にシステム側を記述します。「＜アクタ名＞は～する」、「システムは～する」という書式で、アクタとシステムの間でフォーカスが切り替わるたびに、イベントフローのステップをひとつ進めていきます。

イベントフローには、最も基本的なシナリオを表す「**基本フロー**」と、条件や例外の発生などによる分岐で生じるシナリオを表す「**代替フロー**」とがあります。ユースケースひとつに対して基本フローはひとつですが、代替フローは0個以上定義することができます。

基本フローのステップ数が多すぎる場合は、基本フローの中に代替フローに相当する分岐が含まれている可能性があります。**イベントフローは、分岐が発生する書き方は避けて、単一の流れとなるように記述**します。

基本フロー（悪い例）

1. システムは、登録が受付中かどうか確認する。
2. …以降のステップを記述

基本フロー(良い例)

1. システムは、登録が受付中であることを確認する。
2. …以降のステップを記述

悪い例では、ステップ1の確認結果によって分岐が発生することになります。ひとつのイベントフローの中に複数のシナリオが含まれることになり、確認が必要なシナリオを見落とす危険性が出てきます。

基本フローを定義したら、ステップ毎に、そのステップで分岐が発生する可能性がないか、例外が発生する可能性がないかを確認し、可能性がある場合は代替フローとして記述していきます。代替フローの記述側では、エントリーポイントと終了ポイントを明記します。終了ポイントでは、基本フロー内の特定のステップに戻るのか、終了するのかを明示します。

代替フロー１）スケジュールが見つからない
基本フローのステップ 3 で、利用者のスケジュールを取得できない場合、実行される。

1. システムは、利用者にエラーメッセージを表示する。
2. 利用者は、エラー内容を確認後、基本フローのステップ１に戻る。

イベントフローのステップ数が多すぎる理由は、複数のユースケースがひとつにまとめられているためかもしれません。その場合はユースケースを更に分割する必要があります。イベントフローのステップ数は、**基本フローが10ステップ以内に収まる**ことを目安とし、これを大幅に超える場合はユースケースの粒度の見直しを行うほうが良いでしょう。

GSLetterNeo Vol.57
2013 年 4 月 20 日発行
発行者●株式会社 SRA 産業第 1 事業部
編集者●土屋正人、柳田雅子

バックナンバを公開しています●http://www.sra.co.jp/gsletter
ご感想・お問い合わせはこちらへお願いします●gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋２－３２－８