アジャイルソフトウェア開発向けUML適用
ガイドライン
Ver 1.1

2016年6月

特定非営利活動法人 UMLモデリング推進協議会

アジャイルソフトウェア開発部会

目　　次

1. まえがき

1.1 このガイドラインの対象者

このガイドラインは、UML は使った経験がありアジャイルソフトウェア開発を始めてみたいがきっかけが掴めないチームリーダやマネージャ、あるいは今まで UML やアジャイルソフトウェア開発に興味があったがやり方が分からなかった開発者や、製品をより早く市場に出すことよりも品質を向上させたい開発者に向けて書かれています。

しかし、本ガイドラインを読み進める上で UML 及びアジャイルソフトウェア開発共にそれ程多くの知識を必要とはしません。これらについてより詳しく知りたい読者は、参考書籍を参照して下さい。UML について詳しく知りたい方は参考書籍の(1)が、開発時に於ける UML モデルの使用方法については参考書籍(2)が参考になるでしょう。また、アジャイルソフトウェア開発のプラクティス（アジャイルプラクティス）については(3)が、Scrum については(4)と(5)がそれぞれ参考になるでしょう。更に、優先順位付けの手法としては、参考書籍の(6)により多くの手法が記載されています。

1.2 このガイドラインの読み方

このガイドラインは、今までの書籍にはなかった、UML とアジャイルソフトウェア開発両方の観点から書かれています。これは、開発者間の意識の共有には抽象度の高いモデルを使用した方が有効であったり、モデルを具体的に説明するのにアジャイルプラクティスが有効であったりするためです。

2章ではモデリングの必要性を、3章ではプロダクトバックログ項目を見つける方法や、プロダクトバックログの表し方の一例を紹介すると共にアジャイルプラクティスを使用して説明を付与する方法を紹介しています。更に4章はアジャイルソフトウェア開発を行う上で適用できる Tips 集となっています。

各章はそれぞれ独立した内容となっていますので、2章から順に読んでも、必要な部分から読んでも構いません。

1.3 ガイドラインの背景

昨今、ソフトウェア開発に携わる人々の間で「アジャイルソフトウェア開発」というキーワードがよく聞かれるようになっています。特にアイデアを素早く形にし、市場に提供することがより利益を生むようなサービス業ではこの傾向がより顕著に表れていると思われます。この様な業種では、多少ユーザの使い勝手が悪くても製品を市場に提供するまでの時間を短くし、早期に使用してもらいフィードバックを反映していくことによりユーザにより高い価値を提供することができます。

組み込みソフトウェア業界では、市場に製品をより早く提供することよりも決められた製品サイクル内で安定した品質の製品を製造することが重要であり「アジャイルソフトウェア開発」の需要が低い状況にあると思われます。更に、ソフトウェア開発と比較し、ハードウェア開発には時間が掛るため、製品の開発速度がソフトウェア開発速度に比例しないといった状況もアジャイルソフトウェア開発の需要が低い一因であると思われます。

一方、組み込みソフトウェアの特徴であるイベント駆動型のプログラムの動作は、UML で規定されたステートマシン図を使用することで容易に分かり易く表すことができたり、クラス図によってハード

ウェアとソフトウェアの境界を明確にし、ハードウェアとソフトウェアの結合を疎にすることが容易にできたりします。また、情報家電の様な一般消費者向けの組み込み製品は、商品企画の際に消費者がどの様に製品を使用するのかを想定し、その想定に基づき製品開発をすることが重要です。その際、ソフトウェアのことをよく知らない企画側が想定した情報をできる限り欠落させることなく開発側に伝えることがキーポイントとなります。これには全体の機能を表すことができるユースケース図や操作の流れを表すことができるアクティビティ図の使用が有効です。更にユースケースを開発の単位とするユースケース駆動開発では、モデルを繰り返し検討し洗練しながら品質の高いソフトウェアを製作することができます。このユースケース駆動開発は、組み込みソフトウェアが UML と親和性が高いだけでなく、アジャイルソフトウェア開発の考えを取り入れられることを示しています。なぜなら、ユースケース単位の開発は、Scrum におけるプロダクトバックログアイテム単位の開発に対応させることができるからです。

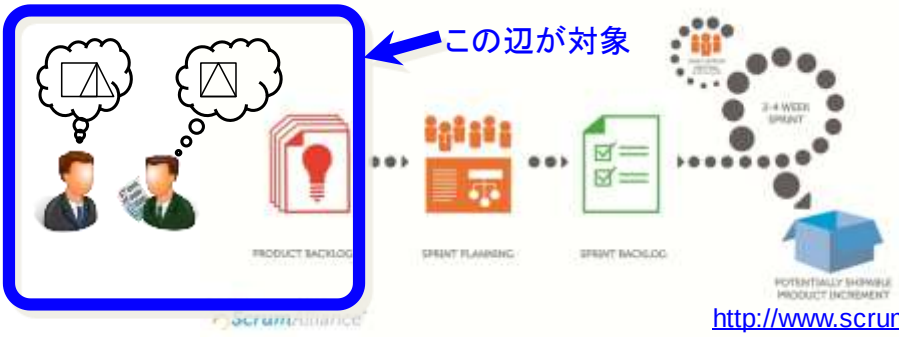
この様に抽象度の高いダイアグラムと具体的な記述とを組み合わせることにより、高いレベルで開発者間の意識共有がされると共に、反復毎に妥当性確認がされスピーディーで確実な開発が行えると考えます。この様な考えは、組み込み製品に限らずシステムの開発全般に於いても適用できると考えます。

この様な背景から本ガイドラインは UML をより活用出来るよう UML 等を使用したモデリングにアジャイルプラクティスを適用する方法を紹介しています。また、アジャイルソフトウェア開発にモデルを適用する方法についても述べています。

なお、本ガイドラインはアジャイルマニフェスト（http://agilemanifesto.org/iso/ja/）として表わされている考えを基にし、ソフトウェア開発手法は Scrum（スクラム）に則っています。

図１－１は、Scrum Alliance が公開している Scrum のフレームワークに本ガイドラインの記述範囲の１つであるプロダクトバックログ項目を洗い出す作業の概念を表したものです。このプロダクトバックログ項目を洗い出す作業を3章で説明しています。

なお、4章の Tips については、図１－１に示した範囲に限らず適用できる内容としています。



http://www.scrumalliance.org/why-scrum

図１－１　本ガイドラインの対象範囲

2. なぜモデリングなのか

2.1 この章の目的

本章では、モデリングを行うに当たり、モデリングのメリットやモデリングツールの要否についての考え方を示しています。

2.2 モデルを使うことのメリット

物事を簡潔に表現するべく、その本質だけを抜き出して記述したものをモデルと言います。モデルを表現するための言語をモデリング言語といいます。モデリング言語は自然言語よりもプログラミング言語に近いため、システム構築やソフトウェア開発に特有の概念や技術を簡潔に表現することができます。

ソフトウェア開発に特化したモデリング言語を用い、モデリングを行うことには次のようなメリットがあります。

(1) どこでも使うことができる

モデリングは特定のフェーズや条件に制限されません。複数の図を組み合わせることで、一つの図では表現できない意図を示すこともできます。要求や業務フローを表現するときはもちろん、情報共有のためのちょっとした会話、実装作業の内容を整理する際など、様々な局面でモデルを使うことができます。

(2) 事前に妥当性を検証することができる

モデルは問題領域や成果物としてのシステムを抽象的に記述したものです。実際にプログラムを作成するよりもはるかに短時間で構築することができます。そのため、事前にモデルを作成しその妥当性を検証することで、実際にプログラムを作成するより短時間で設計・実装の妥当性を確認することができます。

(3) 本質だけを記述できるので、課題に集中することができる

設計について話をしたい場合、実装の詳細は必要ない場合があります。また、要求について話をしたい場合、実装や設計の詳細は必要ありません。構成要素を捨象し単純化することにより、必要以上の詳細に気を取られることなく、課題を整理・議論することができます。

モデリングのための言語に UML(Unified Modeling Language)があります。UML は国際標準化団体 OMG(Object Management Group)が策定している汎用モデリング言語であり、アジャイルマニフェストの共著者である Ivar Jacobson が、James Rumbaugh や Grady Booch と共同で作成しました。ソフトウェア業界では事実上の標準モデリング言語となっています。

OMG http://www.omg.org/

UML 仕様書 http://www.omg.org/spec/UML/

2.3 モデリングツールが必要か

モデリングを行う上で最初に検討すべきは、「モデリングツールが必要かどうか」です。高いアジリティを必要とするサービス業に於いて、開発者間での情報共有にモデルは有効かもしれませんが、モデルを描くのに最も有効なツールはホワイトボードかもしれません。もしかしたら、アイデアを書きとめるためのメモ帳だったり、紙ナプキンだったりするかもしれません。ですから、モデリングツールの必要性は最初に検討すべき項目となります。

しかしながら、本ガイドラインは、「品質を向上させたい開発者」も対象としています。

この様な開発者にとっては、モデルは設計文書そのものとなりえます。また、ドメインエキスパートと開発者間でのコミュニケーション手段として継続的にモデルを使用することでブレークスルーを引き起こし、より分かりやすい正確なモデルへ成長させることができます。この様な開発者にとっては、モデリングツールは必須のツールと言えます。

【オフィス製品はモデリングツールの代わりにはならない】

では、Word®、Excel®または PowerPoint®といったオフィス製品はモデリングツールの代わりとなりえるでしょうか。基本的には、「No」です。描いたモデルを文章化するだけなら使えるかもしれませんが、それだけならホワイトボードに描いたモデルをコピーしたり、メモ帳に描いたモデルをスキャンしたりして電子ファイルにしておけばいいかもしれません。

クラス図やステートマシン図はモデルをある一面からみたスナップショットです。モデリングツールではこれらは有機的に繋がっており、ある変更は影響する他の部分に直ちに反映されます。しかし、オフィス製品で描いたモデルは単なる「絵」であり相互の繋がりは有りません。ある部分を修正したら関連する部分を「人」が全て見つけ出し修正する必要があります。これはコストが掛かるだけでなく修正漏れによる矛盾を引き起こし、バグの元となります。この点だけ見てもオフィス製品はモデリングツールの代替とはなり得ません。

2.4 モデリングツールの選定

最近のモデリングツールには多くの製品が有り、有する機能も様々です。今では多くのモデリングツールがあり、無償、有償の違いだけでなく、機能も様々です。自分達が本当に必要とする機能は何かを見極め、製品を選択する必要が有ります。以下に、選択のポイントを幾つか挙げておきます。但し、これらの項目が検討すべき全ての項目ではありません。環境に合わせ必要であれば更に検討して下さい。

(1) どんな図を描く必要があるか

モデリングツールには、UML だけでなく、SysML やマインドマップに対応したものが有ります。

(2) ソースコードを自動生成する必要があるか

クラス図やステートマシン図からソースコードが生成できれば作業効率と品質が向上します。

(3) 既存のソースコードを利用する必要があるか

　既存のソースコードや購入したライブラリをモデルとして利用する必要がありますか。必要があるなら、リバースエンジニアリングに対応している必要があります。

(4) 要求管理ツールとの連携は必要か

　要求の管理を専用ツールにまかせつつ、タイムリーにモデルに取り込むことができれば、要求とモデルの乖離が減り品質が向上します。

(5) モデルベースでどこまで確認するか

　プログラムの正確さは動作させてみるまで確認できません。モデルを描いた時点で動作を確認することができれば手戻りが少なく作業効率が向上します。

(6) 開発環境との親和性はいいか

　統合開発環境（IDE）やプロジェクト管理ツールとシームレスに繋がることで作業効率が向上したり、作業管理が容易になったりします。

3. モデルとアジャイルプラクティスを有効に使用する

3.1 この章の目的

この章では、モデルとアジャイルプラクティスのそれぞれの欠点を相互に補うことにより、より効果的な成果物を作成する方法を提案しています。

3.2 初期のプロダクトバックログアイテム発見作業

Scrum では、製品が有する機能を順位付けしリスト化しプロダクトバックログとして管理します。このプロダクトバックログの項目であるプロダクトバックログアイテム（Product Backlog Item ： PBI）はプロダクトオーナーが管理責任を持ちますが、受託開発に於いては顧客と共に必要となる機能・性能を要求として見つけ出し PBI としていく作業が必要となる場合があります。しかし、顧客は何が必要であるか自身でも理解していない可能性があり、プロダクトオーナー及び開発チームにはこの顧客の暗黙知の部分を引き出すことが求められます。この作業は一般的に創造的で終わりのない困難を伴う属人的な作業となりがちです。更に、最初に見つけ出された PBI の価値は、ビジネス環境と共に変化していくため、プロダクトオーナーはビジネス価値に従い PBI の優先度を決定したり、追加や削除をしたりしてニーズに合致するよう管理していくことが必要となります。また、開発チームは技術的観点から開発リスクの高い PBI を洗い出し、優先度決定の支援を行います。

図3.2－1にソフトウェア開発全体の流れ及び PBI 発見作業の位置付けを示します。

なお、図中では追加の PBI は「機能の詳細化」及び「実装」時に発見されていますが、他の作業中に発見されることもあります。

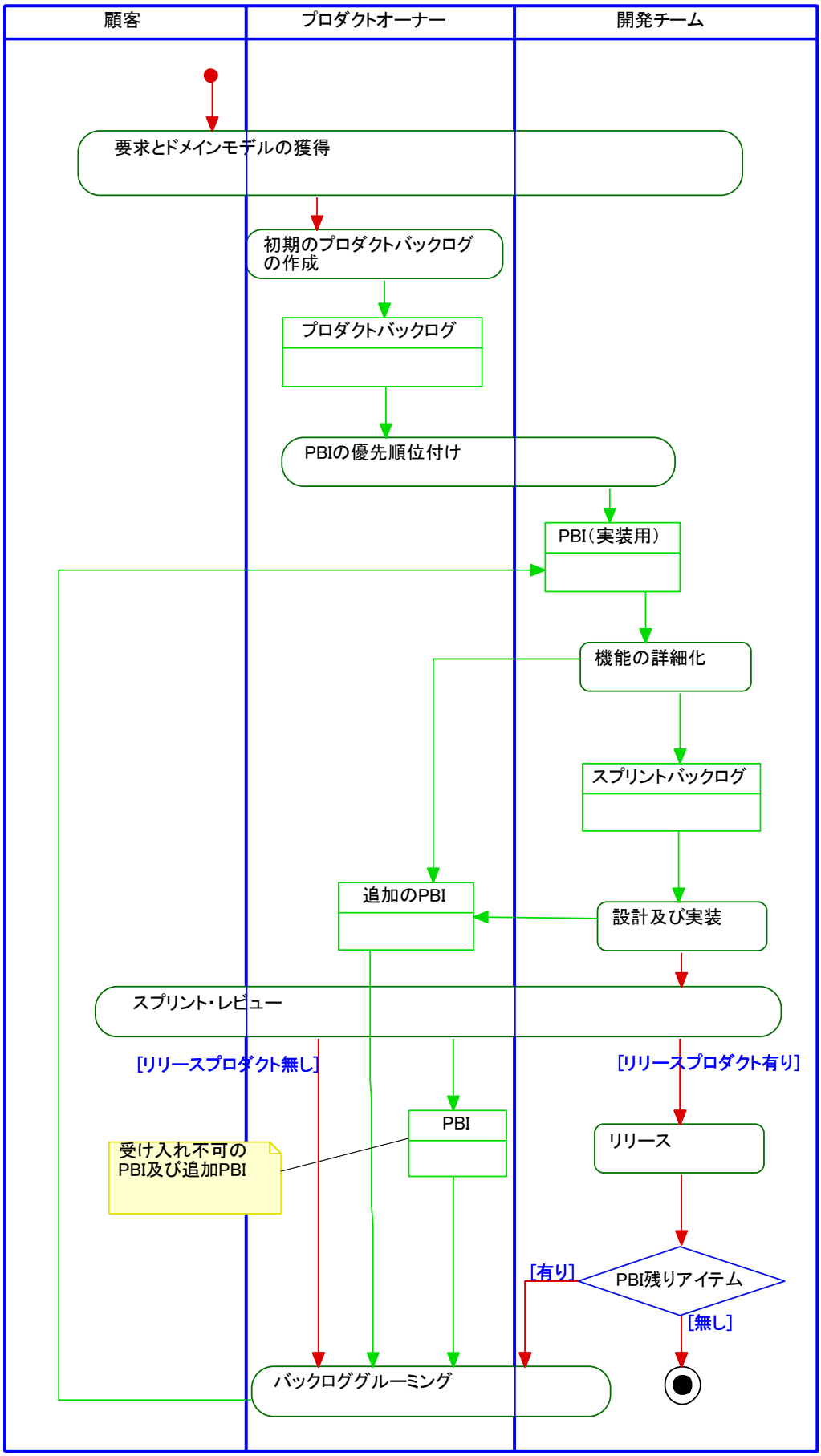


図3.2－1　ソフトウェア開発全体に於ける要求開発の位置付け

3.3 要求とドメインモデルの獲得

3.3.1 インタビューによる機能・性能の洗い出し

（アジャイルソフトウェア開発とモデリングセミナー（2012 年 3 月） 平鍋氏講演資料より）

顧客の暗黙知の部分を引き出す初期段階ではインタビューを行いながら必要な機能を洗い出してくことが有効となります。また、この段階では定型フォーマットやダイアグラムに表わすことよりより多くのアイデアを導き出し洗練していくことが大切となります。しかし、まったく何もない状態からヒアリングを始めると議論が発散するだけでなくヌケ・モレが発生しやすかったり、後工程で使いにくかったりします。

それらを回避するため、ヒアリング時にマインドマップ等、アイデアを見える化できるツールを使用し顧客の頭の中の隠れたアイデアを導きだすことが重要となります。

図3.3－1にユーザ要求聞き取りテンプレートをマインドマップで表わした例を示します。また、表3.3－1に聞き取り項目の説明を示します。

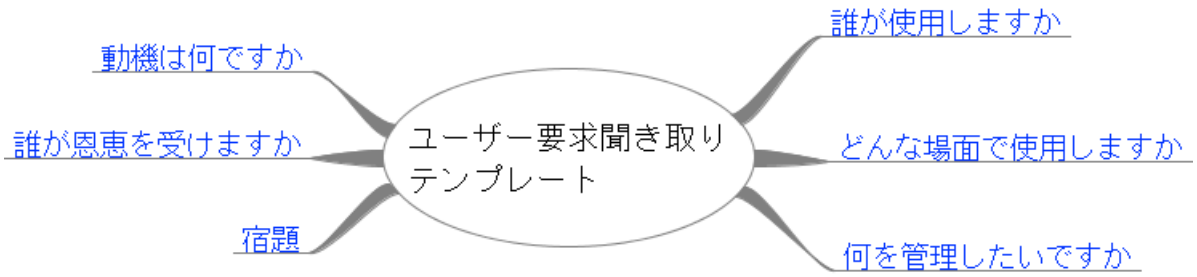


図3.3－1 ユーザ要求聞き取りテンプレート

表3.3－1 聞き取り項目の説明

No	テンプレート項目	説明
1	動機は何ですか	最も基本的な要求の背景や本質を聞き出す質問。
2	誰が恩恵を受けますか	他の重要なステークホルダーを聞き出す質問。この人達にもインタビューの必要があるかも知れない。また、各ステークホルダーが思い描くゴールを聞き出しておくと良い。
3	誰が使用しますか	システムのユーザを聞き出す質問。
4	どんな場面で使用しますか	システムの目的、利用場面を聞き出す質問。
5	何を管理したいですか	システムが扱う概念群を聞き出す質問。
6	宿題	質問について、うまく回答が得られなかった場合、ここにメモを残すようにする。

3.4 初期のプロダクトバックログの作成

顧客へのインタビューで獲得しユースケースとして表わした要求を PBI とし、プロダクトバックログとしてのユースケース図を作成します。作成には以下のプラクティスを使用します。

3.4.1 ユースケース図による要求項目の見える化

前項の「ユーザ要求聞き取りテンプレート」を使用した場合、テンプレート項目と UML のダイアグラム又はアジャイルソフトウェア開発での用語（表し方）との対応は表3.4－1に示す通りとなります。

この段階で見つかったクラス候補は、顧客との共通言語（ユビキタス言語）で表わされることによりドメインモデル作成の基となります。また、これを基にユースケース図を作成し、プロダクトバックログとして管理し、初期の開発範囲（スコープ）を明確にします。このスコープは今後の作業進捗により変化する可能性があります。

表3.4－1 テンプレート項目、UML ダイアグラム及びアジャイルソフトウェア開発での用語（表し方）との対応

No	テンプレート項目	UML ダイアグラム		アジャイルソフトウェア開発での用語（表し方）
		要素	ダイアグラムの種類	
1	動機は何ですか	―	―	―
2	誰が恩恵を受けますか	―	―	ステークホルダー
3	誰が使用しますか	アクター	ユースケース図	ペルソナ
4	どんな場面で使用しますか	ユースケース	ユースケース図	PBI、ユーザーストーリー
5	何を管理したいですか	クラス	クラス図	―
6	宿題	―	―	―

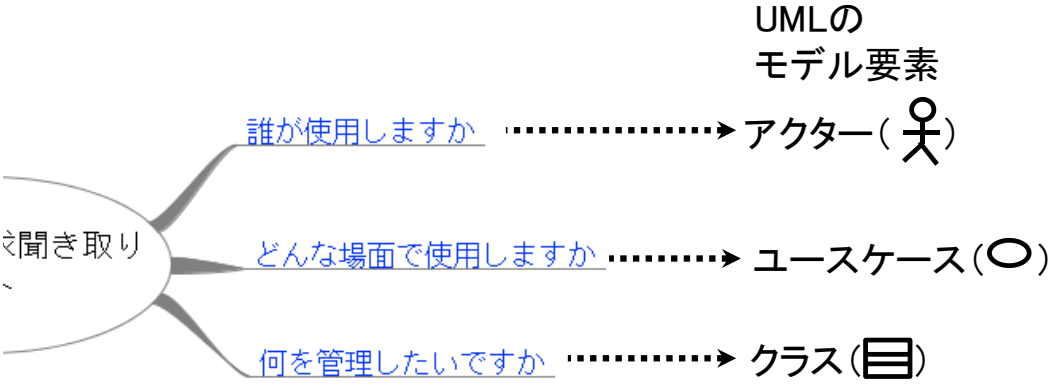


図3.4－1 ユーザ要求聞き取りテンプレートと UML のモデル要素のマッピング

3.4.2 ユースケース図作成時の問題点

ユースケース図を使用したモデリングでは一般的に以下の事柄について注意が必要です。

(1) ユースケースを適切な粒度に保つ。

(2) ユースケース毎にユースケース記述を書く

(3) ユースケース記述の作成に時間を掛け過ぎない。

(4) アクター名は役割にする。

しかし、ユースケースの粒度に関しての定量的な基準は存在せず、また、ユースケース記述を必要以上に詳細化し作成に時間が掛ってしまうことが少なくありません。この様にユースケース図の作成に時間を掛け過ぎるとプログラムの製作に掛けられる時間が少なくなるばかりでなく、ユースケースの削除にためらいが生じます。この様な状況を俗に「ユースケース地獄」と呼びます。

一方、システムバウンダリやアクターはユースケースほど分析されることはなく、分析や定義が不足しがちです。この様な状況は開発対象システムがあいまいだったり、開発対象システムとコラボレーションする人や外部システムについて開発チーム内で合意形成されなかったりして、プロジェクト後半になって問題となる場合が有ります。

なお、図3.4－2にユースケース図の例を示します。

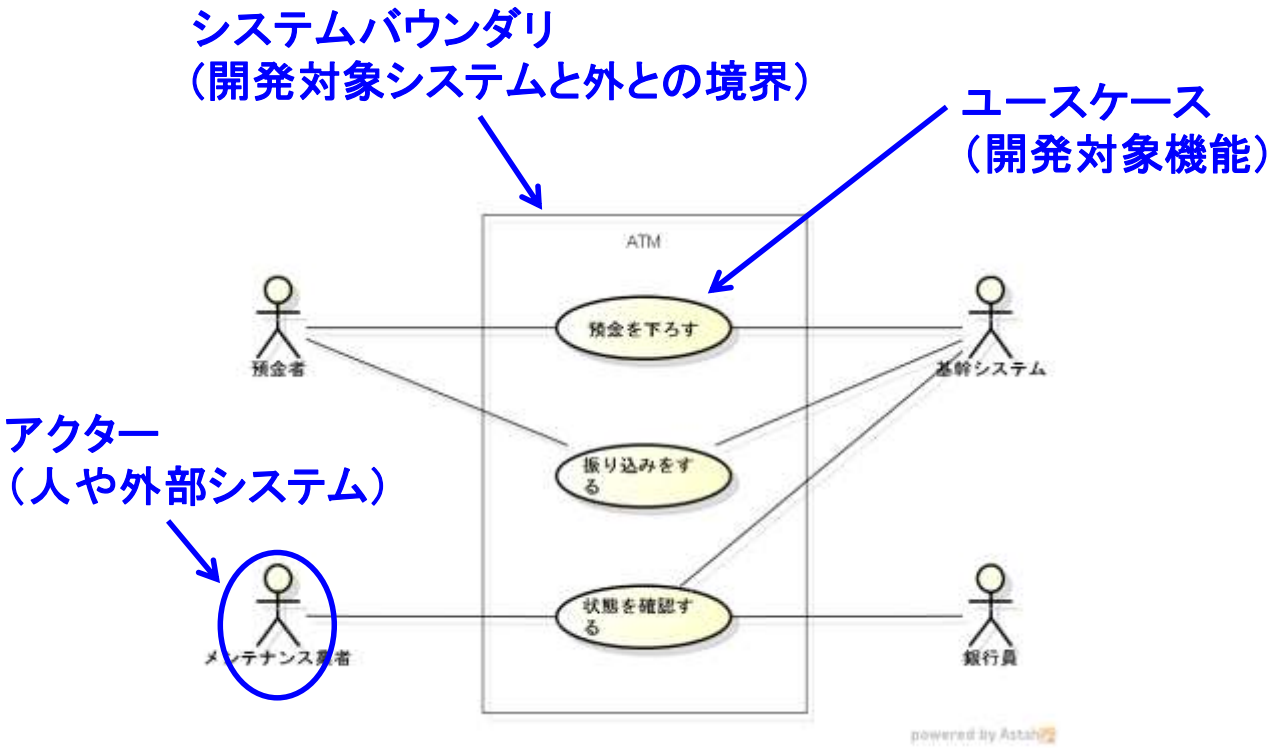


図3.4－2 ユースケース図（例）

3.4.3 ユースケース地獄に対応する

前述したように全てのユースケースに対しユースケース記述を書いていると多くの時間を消費してしまいます。ユースケース記述は開発には必要な文書ですが、書き過ぎるとプロジェクトに弊害をもたらします。

アジャイルソフトウェア開発では、機能に当たるフィーチャーを説明するために「ユーザーストーリー」を使用することが有ります。ユーザーストーリーとは、実現したいと思っているフィーチャーを簡潔に示したものであり短い文章として表します。この様な短い文章はユースケース記述の様に書くのに時間が掛りません。

解決策としてユースケース記述は、ユースケースの実装の順番が回ってきた時にスプリント計画会議等で必要な分だけ作成するようにします。この様にすることでユースケース記述の書き過ぎを防ぐと共に詳細な動作をジャストインタイムで表していきます。

ユーザーストーリーを記述する際に大切となるのはフィーチャーの本質を捉えるキーワードを書き留めておくことであり、フィーチャーのありとあらゆる詳細を書き留めることではありません。ユーザーストーリーは仕様としてはあまりにも短く、あいまいなため不安を覚えますが、後で必要となった時に何を話せば良いのかを思い出すための覚書であるためあいまいさが問題となることは有りません。あいまいさは、後に詳細なユースケース記述を作成するための約束と捉えて下さい。ユーザーストーリーのフォーマット例としては以下のものが有ります。

（例1）「役割」として「結果」が欲しい、それは「理由」のためだ。

（例2）「役割」として「機能や性能」が欲しい、それは「ビジネス価値」のためだ。

良くかけているユーザーストーリーの特徴の1つ目は、顧客にとって何かしらの価値が書かれていることです。ここでの「価値」とは、顧客が対価を払ってもよいと思えるものであり、狩野モデル（3.4.7⑵項参照）を使用して分析されることも有ります。また、ユーザーストーリーはビジネスの観点から評価できなければならないため、顧客との共通言語である「ユビキタス言語」を使用して記述する必要があります。

2 つ目の特徴は、エンド・ツー・エンドとなっていることです。エンド・ツー・エンドとは、例えば顧客の操作から始まって処理が完了するまで、具体例としては、「文字列を選択し斜体文字に変換するアイコンをクリックすると選択された文字が斜体文字となる」までを言います。

更に、よく書けているユーザーストーリーは表3.4－2に示す 6 つの特徴を備えています。また、通常ユースケースの説明に使用するユースケース記述とユーザーストーリーとの比較を表3.4－3に示します。

表3.4－2 ユーザーストーリーの INVEST

No	特徴		説明
1	Independent	独立している	できるだけエンド・ツー・エンドで定義し、ストーリー毎に作成する／しないを決められる様にしておく。
2	Negotiable	交渉の余地がある	予算や期間に合わせて柔軟に実現手段を選択できるようにしておく。
3	Valuable	価値のある	ストーリー単体でも顧客が対価を払っても良いと思えるフィーチャーとする。
4	Estimatable	見積もれる	ストーリーの実装規模を見積もれるようにする。このためにはストーリーはある程度小さくすることが必要。
5	Small	小さい	
6	Testable	テストできる	テストできないストーリーは何を持って「完了」したか分からないのでテストできることは重要。

表3.4－3 ユースケース記述とユーザーストーリーとの比較

項目	ユースケース記述	ユーザーストーリー
作成目的	目的、事前条件、事後条件、シナリオの詳細を表す	ユースケースの本質を捉えたキーワードを残しておき、実装の際のコミュニケーションのきっかけとする
記述内容	ユースケースの概要 事前条件 事後条件 ユースケースシナリオ　等	ユーザ視点から見た実現したいと思っているフィーチャー
記述例	概要 　システムにログインする 事前条件 　・認証システムと正常に通信できること 　・認証システムにアカウントが作成され、パスワードが設定されていること 事後条件 　・システムにログインできること ユースケースシナリオ（正常処理） 　1. ユーザは、アカウント名とパスワードを入力する 　2. システムは、アカウント名とパスワードが正しいことを認証システムに問い合わせる 　3. 認証システムは、アカウント名及びパスワードが正しいことを確認する 　4. 認証システムは、アカウント名とパスワードが正しいことを応答する 　5. システムは、初期画面を表示する 　　：	ユーザとしてシステムにログインするための機能が欲しい、それは個人毎にパーソナライズされた機能を使用するためだ。

3.4.4 アクターのあいまいさに対応する

ユースケース図では製品と相互作用する人や外部システムを抽象化した「アクター」として表し、役割や名称以外を明確にすることはめったにありません。このアクターの内、システムと相互作用する人を具体的に理解できれば、彼／彼女等のニーズに応えられる製品を作れる筈です。

アジャイルソフトウェア開発では、「アクター」の様な「人」を表すために「ペルソナ」を作成することが有ります。

ペルソナとは、製品のユーザを役割毎に簡単に説明したり、典型的な姿として描いたりしたものです。ペルソナは、リアルな人間であり、具体的に解決したい課題を抱えています。リアルな人間ですから、名前も有り、何か責任を持っています。また、作業手順を詳しく知っていたり、コンピュータに詳しかったりとそれぞれ特徴を持っていたりします。

この様にアクターのインスタンスをペルソナで表わし、製品により具体性を持たせると良いでしょう。

なお、図3.4－3ではペルソナとアクターが依存線により関連付けられていますが、これは説明のためでありツールが対応しているわけではありません。

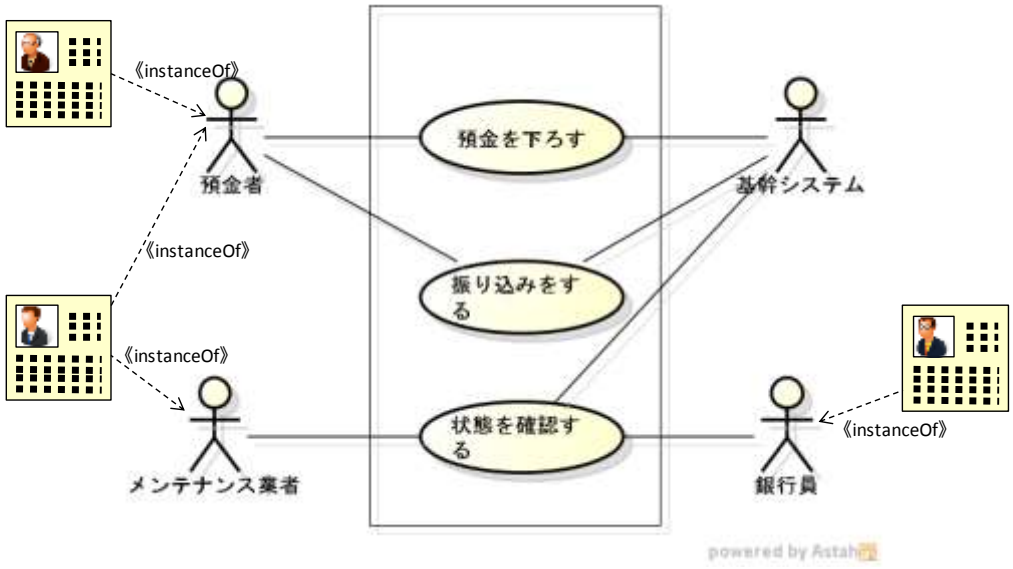


図3.4－3 アクターの説明を追加したユースケース図（例）

3.4.5 開発対象システムのあいまいさに対応する

ユースケース図では開発対象システムを「システムバウンダリ」として表します。しかし、開発対象システムの目的や効果がユースケース図に記載されることはあまりありません。

アジャイルソフトウェア開発では、開発対象システムの目的や効果を「エレベータピッチ」で表わすことが有ります。エレベータピッチとは、ごく短時間でアイデアの本質を伝える手段であり、新規プロジェクトを簡潔に定義するうえでも大いに役立つ手法です。

うまく練られたエレベータピッチには以下のような効能が有ります。

(1) 明快になる

エレベータピッチが目指すのは、全ての人のあらゆる望みに答えようとするあいまいさではなく、製品が何であり、誰のものかを明確にするものです。

(2) 開発チームの意識を顧客に向けさせる

製品は何を提供し、それを提供するのはなぜなのかに意識を向けることで、開発チームは製品の強みが何であり、そもそも顧客が対価を支払うのはなぜなのかを真剣に考えさせることができる様になります。

エレベータピッチを書きあげるための決まったやり方は有りません。以下のテンプレートは、ジェフリー・ムーアの『キャズム』*1 に載っているやり方になります。また、エレベータピッチを作成する際にユーザ要求聞き取りテンプレート項目の「動機は何ですか」の回答を参考にするとより良いエレベータピッチとなります。

エレベータピッチのテンプレート
・「潜在的なニーズを満たしたり、抱えている課題を解決したり」したい ・「対象顧客」向けの、 ・「製品名」という製品は、 ・「製品のカテゴリー」である。 ・これは「重要な利点、対価に見合う説得力のある理由」ができ、 ・「代替手段の最右翼」とは違って、 ・「差別化の決定的な特徴」が備わっている。

・「潜在的なニーズを満たしたり、抱えている課題を解決したり」したい

→顧客が解決したい課題やニーズを説明します。

・「対象顧客」向けの、

→誰のためのプロジェクトなのか、あるいはソフトウェアを使用することで得をするのは誰なのかを説明します。

・「製品名」という製品は、

→名前を付けることで製品は命を吹き込まれます。意図を伝えるという意味でも名前付けは重要です。

・「製品のカテゴリー」である。

→サービスや製品が実態としては何であり、何をするものなのかを説明します。これは「重要な利点、対価に見合う説得力のある理由」ができ、そもそもなぜ顧客がこの製品に対価を支払いたいと思うのかを説明します。

・「代替手段の最右翼」とは違って、

→なぜ既に存在しているものを選択しないのか、その理由を補足します。

・「差別化の決定的な特徴」が備わっている。

→自分達のサービスが競合相手と何が違うのか、より良い部分はどこかを説明し、差別化します。

これは極めて重要です。なぜなら、自分達のプロジェクトへの投資を正当化できる箇所は実質的にはここだけだからです。以下に、エレベータピッチの例を示します。

スーパーの来店客のレジ待ち時間のストレスを楽しみやメリットに変えたい
スーパーマーケット向けの
「待ち時間気にならない君」という製品は、
POS システムの拡張です。
これは、顧客満足の向上と機会損失の低減ができ、
一般的なポイント付与とは違って、
顧客のストレスをメリットに変えることができます。

*1 Geoffrey A. Moore. Crossing the Chasm . Harper Business, 1991.

日本語版:ジェフリー・ムーア著、川また政治訳『キャズム』(翔泳社、2002 年)

3.4.6 ユースケースの規模を見積る

UML の入門書等では「ユースケースの粒度はできるだけ合わせましょう」とされています。これは、ユースケースの粒度が異なると開発規模の見積りが困難となり精度の高い見積りができないためです。

しかし、アジャイルソフトウェア開発では、開発規模を〇〇人・月の様な絶対値では見積らず PBI 毎に相対サイズで見積ります。相対サイズでの見積もりとは、「A」というPBIを基準とした場合、「B」という PBI は「A」の何倍くらいかを見積ることです。そして、見積もりの単位は「人・月」の様な絶対値ではなく、多くの場合「ポイント」として表します。この見積りにあたって使用する相対サイズの範囲は 1～8 に留める様にします。相対サイズが 13 以上になる様なユースケースはとりあえず「エピック」として識別しておき、必要な時期が来たら見積もり可能なサイズに分割し、再見積もりを行います。

この様に、相対見積もりを使用することでユースケースの粒度を揃えることを気にする必要がなくなり、ユースケースの分析に余分な時間を取られることがなくなります。

以下にプランニングポーカーを使った標準的な相対見積もりのやり方を示します。見積りは開発チーム全員で行います。

(1) 基準となる小さいストーリーを開発チームで選択し、そのサイズを”2”とする。

(2) 上記ストーリーの 2.5 倍程度のストーリーを見つけ、そのサイズを”5”とし、これら 2 つのストーリーを基準のストーリーとする。

(3) 見積もり対象の PBI を選択する。

(4) 各自が見積もり対象のストーリーが基準のストーリーの何倍程度かを見積もる。

(5) その数字のカードを伏せて出す。この時、カードは 1 枚とすること。(1 と3のカードを出して、見積もりサイズを”4”とすることは禁止。3 または 5 とすること。)

(6) 開発チーム全員がカードを出し終えたら一斉にカードを裏返して見積もりサイズを確認する。

(7) カードの数値が全員一致していれば、その値を PBI のサイズとする。

(8) 一致していなければ、最大値と最小値を出した人がその根拠を説明する。

(9) 再度各自でサイズを見積もる。

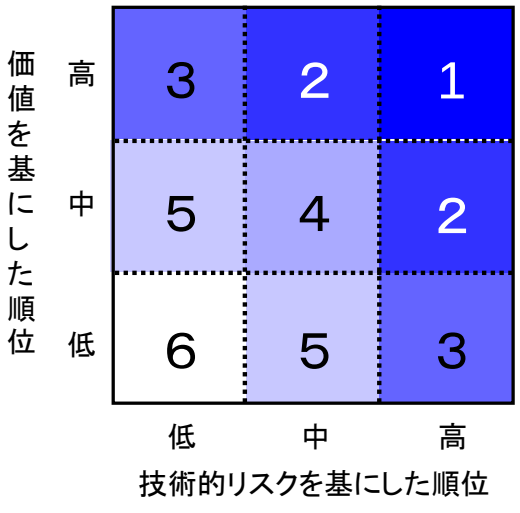
(10) (5)～(8)を繰り返す。3 回程繰り返してもサイズが一致しない場合、最大値を PBI のサイズとする。

(11) 全ての PBI について見積もる。

3.4.7 ユースケースに優先順位を付ける

Scrum では PBI に付けられた一意の優先順位に従い PBI の実装を進めていきます。この優先順位は市場の動向、ROI 及び技術的リスクを基に決定しますが、優先順位を決定するのはプロダクトオーナーの責任となります。開発チームは、技術的リスクについて評価しプロダクトオーナーをサポートします。

簡易的な優先順位の決定方法としては、図3.4-4に示す価値－技術リスクマップにユースケースをマッピングし、優先度の高いエリアに配置されたユースケースから優先順位付けする方法があります。



マスの中の数字は、優先度を表します。
数値が小さい方が優先度高を表します。

図3.4-4 価値－技術リスクマップ

使用する「価値」の評価方法にはいろいろ手法が有りますので、いくつか紹介します。

(1) ユーザーストーリーマッピングを使用する

ユーザーストーリーマッピングとは、図3.4－5に示す通り PBI をユーザが体験するタスクの順番毎（ワークフロー）にマッピングし開発の順位を決定する手法です。優先順位は、実用最小限の製品（Most Viable Product ： MVP）となる PBI を対象として中心となる PBI から放射状に決定していき、その後、他の PBI の優先順位を決定していきます。

本手法は、Jeff Patton 氏（http://www.agileproductdesign.com/）によって開発されました。

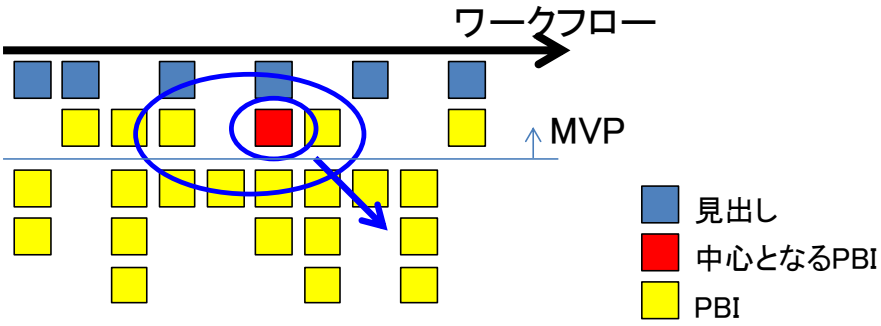


図3.4－5 ユーザーストーリーマッピングイメージ

(2) 狩野モデルを使用する

狩野モデルとは、ユースケースを次の3つのカテゴリーに分類し、優先順位を決定する手法です。ユースケースの量に対する製品の満足度の関係を図3.4－6に、それぞれの説明を以下に示します。

(a) 当たり前、または必須のユースケース

製品が成功するために欠かせないユースケース。必須ユースケースとも呼ばれる。ユースケースをどれだけ幅広く用意しても、品質に磨きをかけても、それが当たり前のユースケースであれば顧客満足度にほとんど影響を与えない。

(b) 線形、一元的なユースケース

「あればあるほど良い」と説明できるユースケース。「線形」または「一元的」という呼び名は、ユースケースの量に対して顧客満足度が線形に高まることに由来する。ユースケースがうまく動けば動くほど、あるいは多ければ多いほど顧客満足度は向上する。

(c) 魅力的な、わくわくするユースケース

大きな満足をもたらしたり、わくわくする気持ちになったりするユースケース。このユースケースがあれば製品の価値を割り増すことができる。しかし、このユースケースが無いからといって顧客満足度が低くなることもない。

以上のカテゴリーを基にユースケースの価値を考えると以下の様になります。

製品が市場で受け入れられるには当たり前のユースケースが必須なので、当たり前のユースケースはすべて備える様に順位を決定する必要があります。しかし、これは必ずしも当たり前のユースケースを早期のスプリントで開発しなければならないという意味ではありません。リリースまでに開発すれば問題はありません。当たり前のユースケースを揃えたら次は線形のユースケースをできるだけ多く揃えることに重点を置きます。そして、魅力的なユースケースは、上記のユースケースを実装した後に、時間の許す範囲で対応するという優先順位にしておきます。

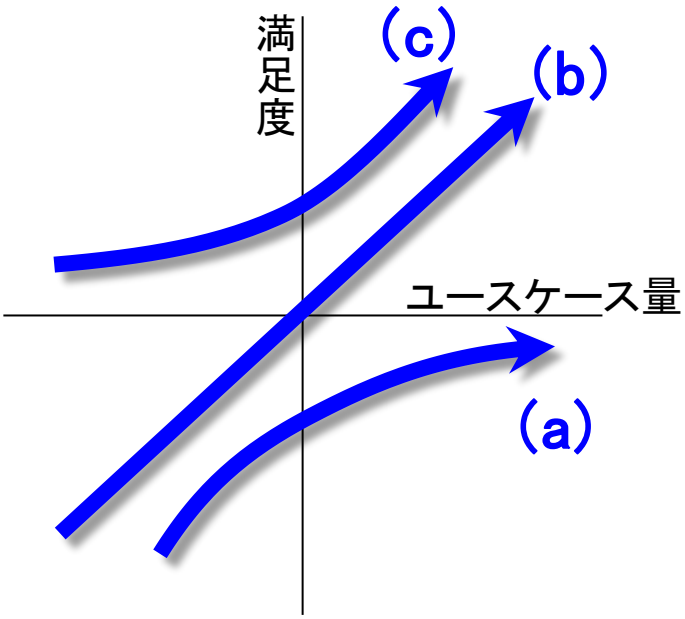


図3.4－6　ユースケース量と満足度の関係

(3) ユースケース図を使用する

本来、PBI は互いに独立しているのが理想ですが全て独立した PBI だけで製品が作られることはほとんどありません。ユースケース図を使用すると図3.4－7に示す通り PBI 間の関連が明確になります。

PBI_1 が PBI_2 をインクルードしている場合、PBI_1 を完了するためには PBI_2 が必要であることが分かります。従って、PBI_2 は PBI_1 より優先順位を高くする必要があります。

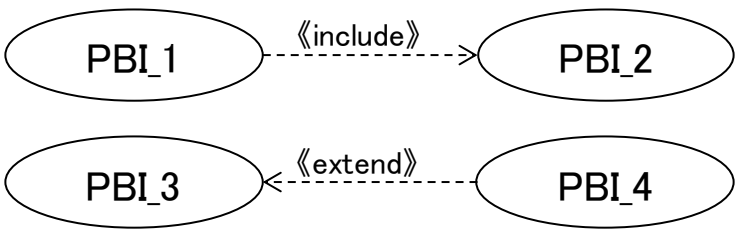


図3.4－7　PBI 間の依存関係

3.5 モデルを使用したPBIの詳細化

3.5.1 シナリオを作成する

ユースケースとして表わされる機能は前述したとおり、自然言語による事前条件、事後条件及びシナリオ等により詳細が明確にされます。しかし、ここではシナリオの記述方法の1例としてアクティビティ図を使用する方法について示します。

アクティビティ図とは、フローチャートに似た図で処理の流れを記述します。企業等では業務の流れを表す際によく似た図を使用します。これらアクティビティ図、業務フロー図どちらもシステムや組織を表すレーンを持ち、処理を担当システムのレーン内に配置していくことで処理の流れを表します。また、条件判断や繰り返し処理も定型の記号により表わすことができるため、全体の可読性が高まります。従って、システム開発に直接係わらない顧客やビジネス側の担当者であっても容易に内容を理解することができます。特にビジネス側の担当者は業務フローに詳しく、コミュニケーションが容易となります。

この様な背景から、PBIの詳細を説明したり、分析したりする際にアクティビティ図を使用することは有効な手段と言えます。

以下にアクティビティ図を記述する流れを示します。

(1) システム及び記述対象のユースケースに関連するアクターをレーンとして表わす。この際、左から順にプライマリアクター（処理を起動するアクター）、システム、及びセカンダリアクター（ユースケースから呼び出されるアクター）の順に並べると良い。

(2) 順次アクションを追加し、代替処理や例外処理を含まない基本となる処理の流れを記述する。

(3) 必要に応じてオブジェクトフローを追加し、データの流れを明確にする。

(4) 基本処理の流れの中から条件判断を含む処理を識別し、代替処理または例外処理を追加していく。

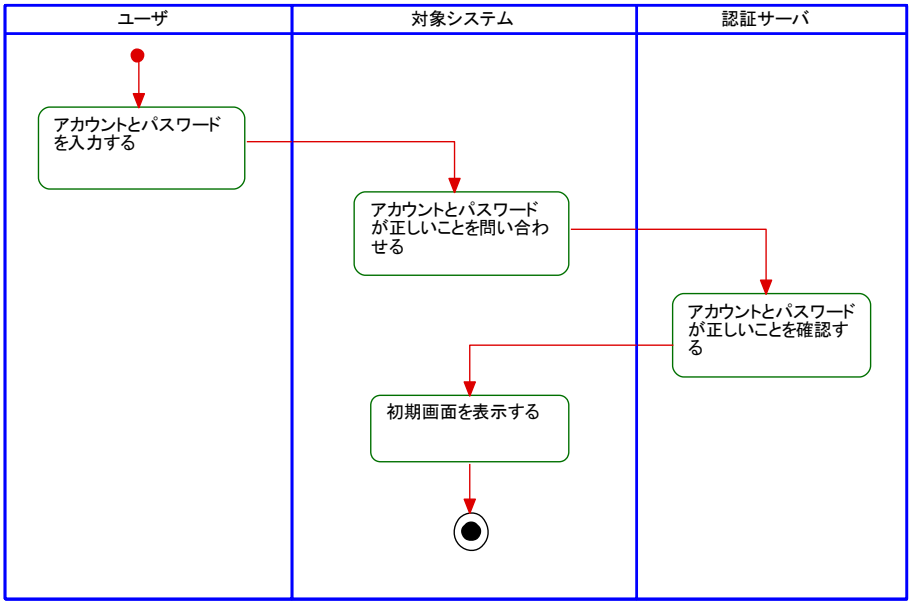


図3.5－1 基本処理だけの初期アクティビティ図（例）

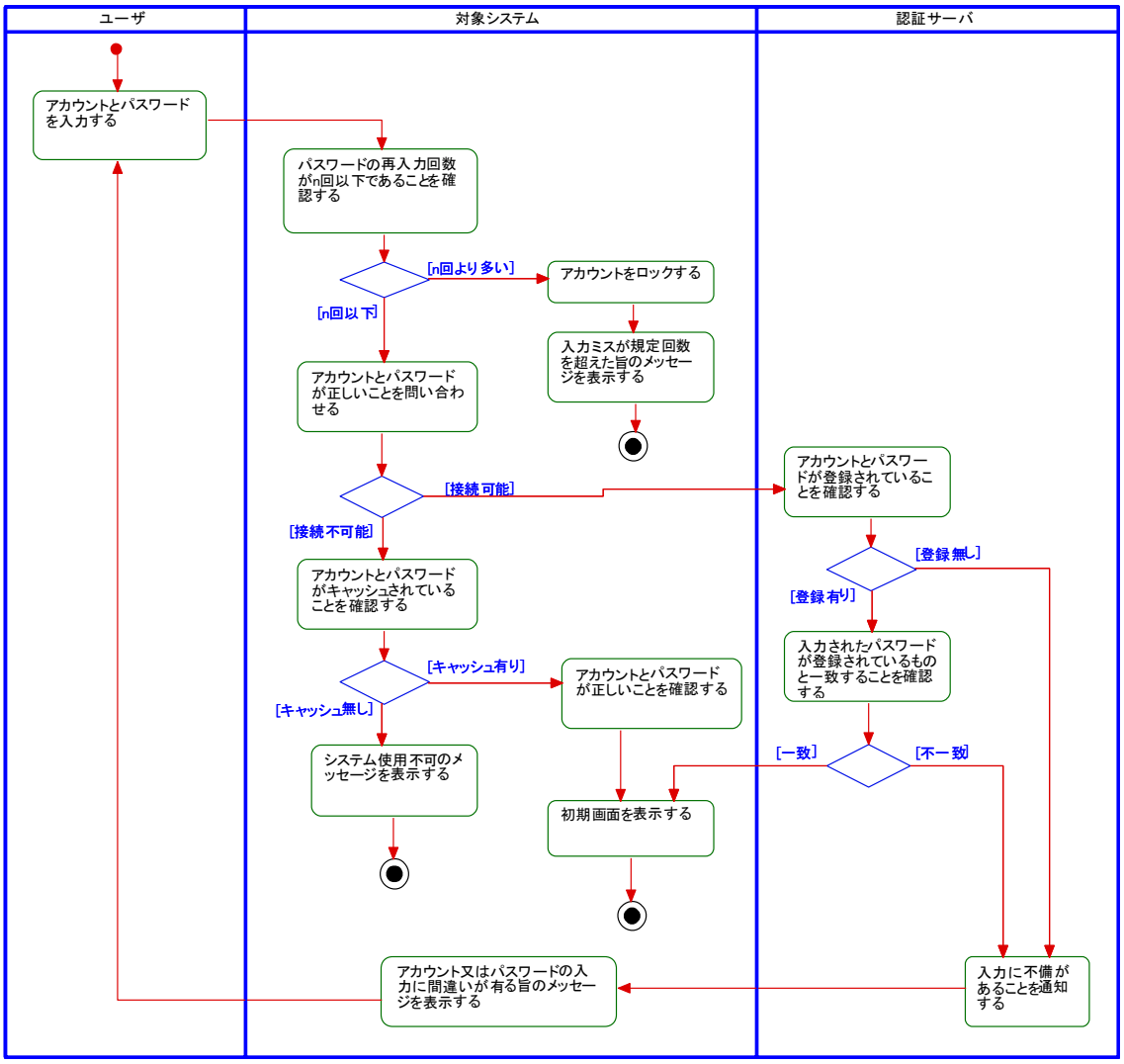


図3.5－2 代替処理及び例外処理を加えたアクティビティ図（例）

3.5.2 アクティビティ図から要求を詳細化する（オプション）

アクティビティ図のパーティション内に描かれた各アクションは、当該パーティションが実行しなければならない内容となり、これらアクションを洗練し要求することができます。これらは、詳細化された要求として捉えることもできますし、サブシステムまたはコンポーネントへの要求と捉えることもできます。

一方、システムを構成するサブシステム候補や、ソフトウェアコンポーネントは業務知識等から導出します。

なお、アクションを洗練し、要求とするには、語尾を「～できること」の様に変更します。

表3.5－1にアクションと要求の対応（例）を示します。

表3.5－1 アクションと要求の対応

No	アクション	要求
1	パスワードの再入力回数が n 回以下であることを確認する	パスワードの再入力回数が n 回以下であることを確認できること
2	アカウントとパスワードが正しいことを問い合わせる	アカウントとパスワードが正しいことを問い合わせられること
3	初期画面を表示する	初期画面を表示できること
4	アカウントまたはパスワードの入力に間違いが有る旨のメッセージを表示する	アカウントまたはパスワードの入力に間違いが有る旨のメッセージを表示できること

変更した部分を下線で示す。

4. アジャイルソフトウェア開発のための Tips

4.1 この章の目的

アジャイルソフトウェア開発では、プロセスや方法論がソフトウェア開発のすべてをフォローしているわけではありません。アジャイルソフトウェア開発の現場では、自分達で考え、行動していくことが求められます。

本章は、部会メンバーがアジャイルソフトェア開発の現場から持ち帰ったものから、普遍性がありそうなものを Tips として部会でまとめたものです。

4.2 Tips の構成

Tips は次のような構成になっています。

名称	Tips の名称です。
目的	その Tips を利用する目的です。
詳細	その Tips の具体的な説明です。

4.3 Tips の使い方

(1) 何かあったら見てみる

アジャイルなプロセスやプラクティスを採用してみたものの、何か違和感があることがあります。それは、何かがかみ合っていないという感覚かもしれないし、とりあえず回っているように見えるけど、何かが足りないという感じかもしれません。そのような時、この Tips を眺めてみて、何か使えたり、参考になったりするものがないか探してみてください。

使えそうな Tips があれば、どのように自分の現場に合わせるのかを考えます。何か工夫をする必要があるかもしれませんし、そのまま使えるものもあるかもしれません。

(2) 全てを使う必要はない。問題解決に使えそうなものだけを使う

アジャイルには絶対となる正解はありません。現場や開発チームが直面している状況に応じて、できることは変わるし、必要となるアクションも異なります。ここにある Tips はあくまで“ある局面において“アジャイルなソフトウェア開発のために使うためのものです。

自分の現場に照らしてみて、問題解決に必要なものだけを利用するようにしてください。

(3) 1つの問題に 1 つの Tips とは限らない。組み合わせて解決策を作り出す

Tips を組み合わせることで、問題が解決するのであれば、複数の Tips を組み合わせてください。このガイドラインに掲載されている Tips は、自分達の現場で見つけた Tips と組み合わせて利用することもできます。

4.4 Tips一覧

【プロセス】

・タスク及びその進捗を壁に貼り出す

・中長期的な品質や保守性に拘る

・必要になるまで作らない

・明確なゴールを決める

【モデリング】

・ふせんモデリング

・ユーザーストーリーから半径1mのモデリング

・ポンチ絵のようなモデル

・合意形成モデリング

・外部文書とモデルのトレーサビリティを確立する

【チーム】

・チャレンジできる環境をつくる

～スーパーマン一人だけでなく開発チーム全員で～

・プロダクトオーナーの意図をこまめに確認する

4.4.1 プロセス

タスク及びその進捗を壁に貼り出す

【目的】

誰の目にもつくところに進捗を貼ることで、問題を早期発見できるようにするため

【詳細】

朝会で進捗を共有しているが、全体としてどうなっているかがよく見えないことがあります。そうなると、スプリントの最後に、朝会では見えてこないようなトラブルが発生しかねません。

開発チームとしての進捗を共有できるツール（かんばんなど）を部屋の目立つ場所に貼り出すことで、何をやっているかが明確になります。全員が現状を把握できるようになるため、開発チームメンバーの仕事を手伝いやすくなります。

もし利用できる壁やホワイトボードがない場合、オンラインの情報共有ツールを利用することもできます。

中長期的な品質や保守性に拘る

【目的】

プロダクトのライフサイクル全体で見たときのライフサイクルコストを小さくするため

【詳細】

リリースを最優先するための間に合わせのコードが増えていくと、技術的負債は少しずつ増えていきます。下記に記したプラクティスをより効果的に利用することで、中長期的な品質や保守性の実現可能性が高くなります。

※参考 ： アジャイルソフトウェア開発で主に推薦されているプラクティス

・動くコードであっても積極的に書きかえる（リファクタリング）

・コードより先にテストを書く（テストファースト開発）

・ビルドとテストを1つのプロセスにまとめる（継続的インテグレーション）

必要になるまで作らない

【目的】

開発の中に潜んでいる無駄を省くため

【詳細】

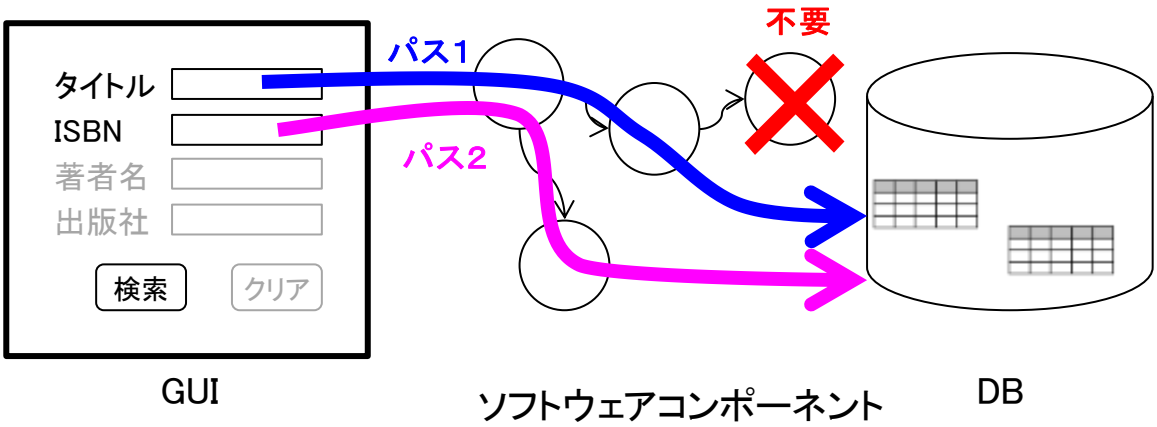
作ることによって得られる知見（設計以前ではわからなかった潜在的な良し悪しが、実際にコードを書いて、テスト完了後、稼働してみて初めて判明する）があるので、図に示す通り少しずつ作る（1 パス通す）ことによって、これらの知見が初期の段階で得られ、学習効果が高くなります。その結果それ以降に作るものに、フィードバックすることが可能になり無駄がなくなります。

得られる知見の例

・設計書のレベルで誤解をまねきやすい書き方が、コードを書こうとすることで判明する。

・いままで当たり前と思っていたロジックが、実はダメなロジックであった。

・テスト性を考慮したコードになっていないことが判明する。



明確なゴールを決める

【目的】

開発チームが無駄なく品質の高い作業を行えるようにするため

【詳細】

1つのフィーチャーを作った時の明確なゴールを決めることが重要です。何をもって完了とみなすか？（単体テスト完了まで？ユーザの受け入れテスト完了まで？マニュアル作成完了まで？など）明確なゴールが決まっていることでやらなければならない作業を詳細に洗い出すことができます。

4.4.2 モデリング

ふせんモデリング

【目的】

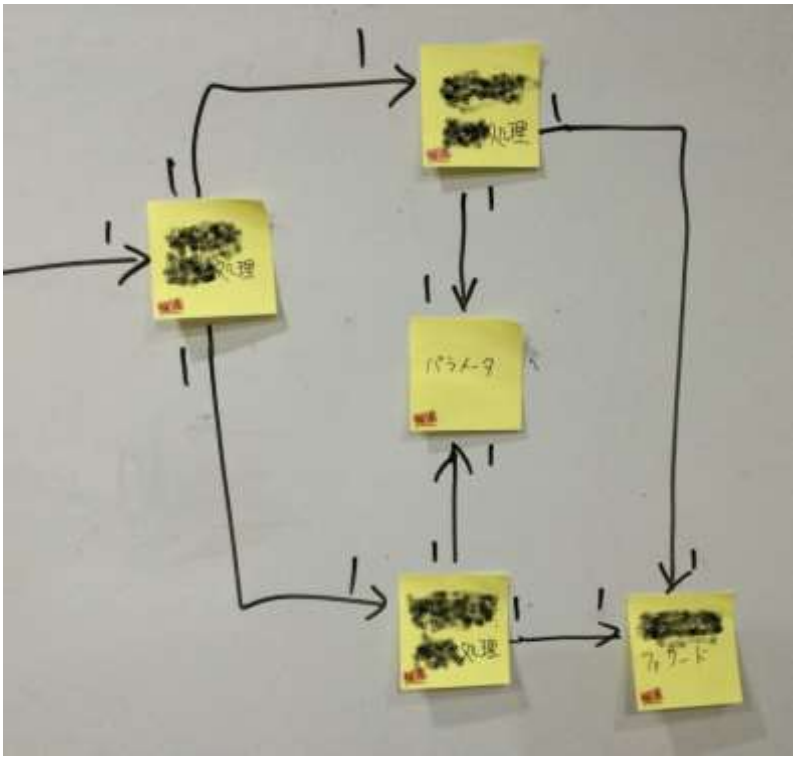
開発の初期段階において、クラス名などがまだ固まっていないときに、複数名で、対象領域の概念およびその関係を明らかにするため。

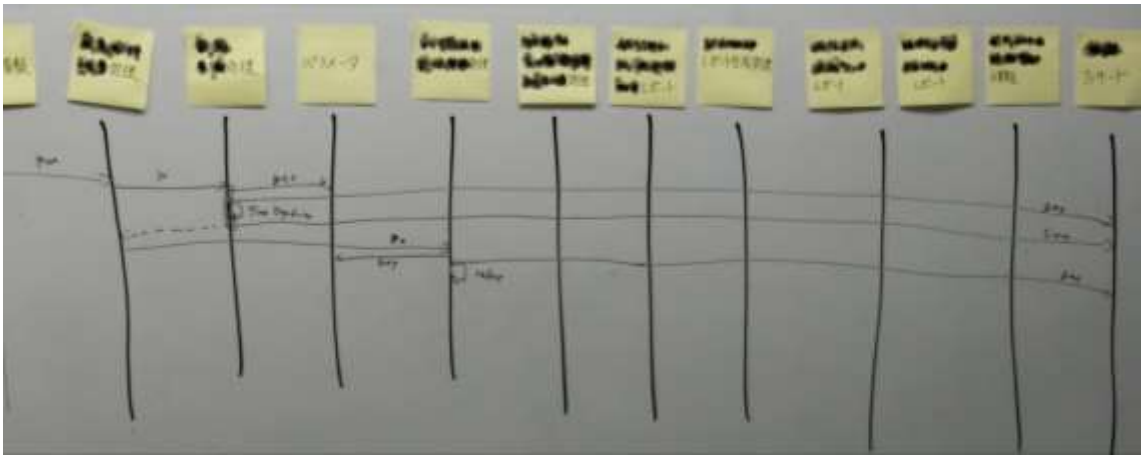
【詳細】

ふせんを追加したり削除したりしながら対象領域の概念を明らかにして、ふせんをクラスやオブジェクトに見立てて、それらの関係を明らかにします。ふせんは動かすことが可能なため、常にふせんの位置関係を変化させながら、あるべきモデルを検討することが容易にできます。

ふせんは模造紙やホワイトボード上に配置します。模造紙で行った場合はそのまま保存することができます。ホワイトボードの場合は、デジカメで結果を残します。

ファシリテイターが参加者の意見をまとめつつ、参加者はそれぞれ、ふせんをはっていきます。それにより全員参加で議論ができます。





ユーザーストーリーから半径1mのモデリング

【目的】

対象の要件に関係するモデル要素だけを集めたモデル図（ビュー）を使用することで関係者の理解を深めるため。

【詳細】

対象の要件に直接関係しないモデル要素を含むすべてを表示した大きくて複雑なモデルを使ってしまうと、理解が困難な場合があります。そのため、関係者が容易に理解できるように、対象を絞り込んだモデル要素を集めたモデル図（ビュー）を使うようにします。

ポンチ絵のようなモデル

【目的】

ラフなモデルを使って、実装する対象の構造や振る舞いを複数人で共有する。

【詳細】

モデルを通してシステムの構造を明らかにし、コードでは表現しきれない意図や考えを伝えていくことができます。

また、どこまで理解できているのか、どこに分からないことがあるのか、そういったことを把握するためにモデルを利用することができます。「分かっている」と思っていることであっても、改めて描き出すことで、理解できていない部分や曖昧な部分が明確になることがあります。この段階では、モデルの完全性は必ずしも重要ではありません。

合意形成モデリング

【目的】

メンバー全員で、モデリング対象の共通理解を持つため。

【詳細】

プロジェクト初期にモデリング能力が高い個人が一人だけでモデルをつくりがちですが、その場合、モデリング対象の見方が偏っていたり、他の人の理解が不十分だったりします。

たたき台のモデルに対して議論をすると当初のモデルの影響を受けやすいので、必要なメンバー全員※1で、その場で共通理解を持つために、全員で議論しながらモデルを一からつくりあげるのが有効です。その際ホワイトボードを利用したり、モデリングツールの入ったパソコンの画面をプロジェクターで映したりします。

※1　開発者に加え、製品に対する責任を負う人なども含みます。

外部文書とモデルのトレーサビリティを確立する

【目的】

モデルだけでは表せない要求の詳細や別途検討した設計の詳細またはテスト手順とモデルを関連付けてモデルの確かさを高めるため。

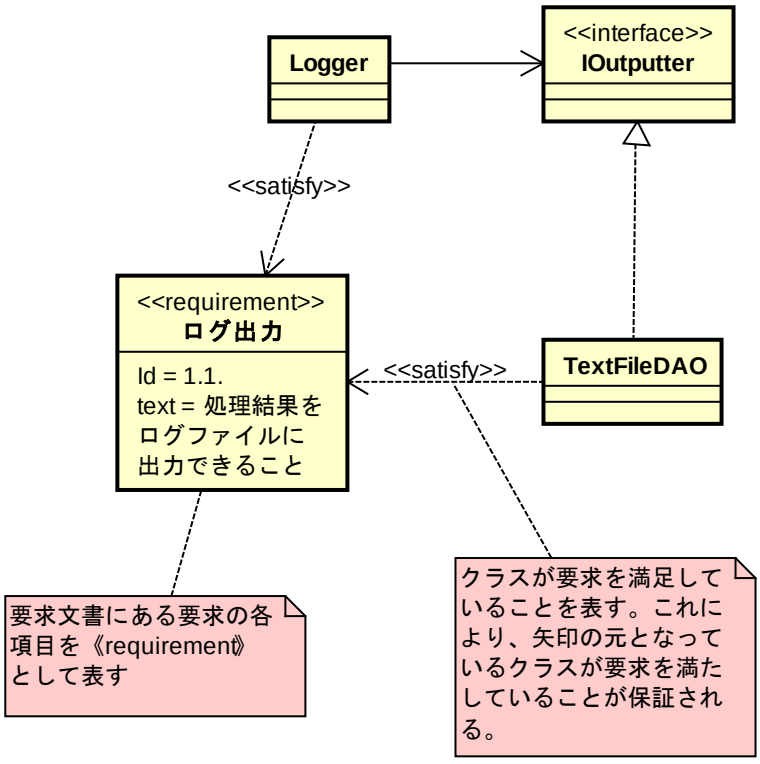
【詳細】

UML のモデルが表す構造や処理内容は対象ソフトウェアのある側面を表したにすぎません。要求項目や処理の流れはユースケース図やアクティビティ図で表すことができますが、それらを要求管理ツールやテスト管理ツールといった専用ツールで管理した方が適切に管理できるでしょう。また、処理の検討結果は技術文書としてまとめられ、管理されることもあるでしょう。

これらツールや技術文書を個別に管理するよりモデルを中心としてそれぞれを関連付けることで要求からテストまでを一貫して管理することができます。

具体的には、各要求を《requirement》としてモデル要素に表し、ユースケースやクラス、アクティビティ図のアクションに関連付けることでモデルとのトレーサビリティを確立することができます。更に、モデリングツールの機能を使ってユースケースやクラスの下に外部ファイルへのリンクや外部システムの URL をインポートすることで技術文書やテスト管理ツールとモデルとのトレーサビリティを確立することができます。

この様にトレーサビリティを確立すると要求がソフトウェアのどこでどの様に実現されるか、またどの様にテストされるか見える化することができ全体の品質向上に繋げられます。



4.4.3 チーム

チャレンジできる環境をつくる
～スーパーマン一人だけでなく開発チーム全員で～

【目的】

スキルアップ、スキルの平準化

【詳細】

リーダーは、できる仕事をできる人にアサインするのではなく、メンバーが自律的に自分のスキル以上の仕事をするように仕向けます。

一人のスーパーマンが自らやりきるのではなく、スーパーマンは、他のスキルが不足しているメンバーを手伝うことで、スキルが不足しているメンバーは自ら難しい仕事に挑戦するような雰囲気をつくります。こうすることで、開発チームの個々のメンバーのスキルアップにつながり結果として、開発チーム全体の生産性が向上します。

プロダクトオーナーの意図をこまめに確認する

【目的】

プロダクトオーナーと開発チームメンバーの認識違いによる手戻りを回避するため

【詳細】

手戻りが発生するのは、認識の齟齬が主な原因です。当たり前だと思えるようなことでも頻繁に確認を行い、プロダクトオーナーの意図と開発チームの意図を細かく擦り合わせるようにします。

手戻りが発生した場合に無駄になるのは時間だけではありません。少ない残り時間で作業を終わらせる必要が生じるため、品質やエンジニアのモチベーションに悪い影響を与える可能性があります。

5. 用語

表5－1　用語集（1／2）

No	用語	定義
1	顧客	開発チームからみた顧客を言い、ユーザだけでなく、社内の要求元も含む。
2	事後条件	ユースケースとして表わされた処理が完了した際に達成すべき条件
3	事前条件	ユースケースとして表わされた処理を行う際に整っていなければならない条件
4	スプリント	開発のタイムボックス。スプリント内で設計、製造、及びテストを実施する。
5	スプリントバックログ	スプリントで実施するタスクの一覧。
6	正常処理	判定処理に於いて全てが「true」の場合の処理の流れ。
7	代替処理	判定処理に於いて、幾つか「false」となるが、やり直し等により最終的には事後条件を満たして終了する処理の流れ。 例：パスワードの入力に於いて、入力ミスをしたが再入力により正常に認証された。
8	バックロググルーミング	PBI の追加や削除を行ったり、優先順位を見直したりしてプロダクトバックログを整理する作業。
9	フィーチャー	システムまたはプログラムが有するエンド・ツー・エンドの機能。
10	プランニングポーカー	相対見積もりをする際に使用するトランプの様なカード。 値は、1、2、3、5、8・・・のフィボナッチ数列になっている。
11	ブレークスルー	ある時を境により良いモデルが見つかること。今までのモデルをベースとしているが、大幅な変更が必要で修正に時間が掛かる場合が多い。しかし、より洗練されたモデルへと成長するため得られるものも大きい。
12	プロダクトオーナー（PO）	プロダクトに責任を持ち、フィーチャーの決定や PBI の優先順位付け等を行う。
13	プロダクトバックログ（PB）	開発の優先順位付けがなされた開発すべきフィーチャーの一覧。

表５－１　用語集（2／2）

No	用語	定義
14	モデリングツール	モデルを描くためのツール。最近では、モデルを描くだけでなく、ソースコードを生成する機能やシミュレーション機能を有するツールもある。
15	モデル駆動開発	システムやプログラムを開発する際に SysML や UML といった標準化されたモデル等を使用し開発を行う手法。
16	ユビキタス言語	ビジネス側（ドメインエキスパート）と開発者との間で共通して理解できる共通語
17	ユースケースシナリオ	処理の流れをアクターとシステムのインタラクションとして記述したもの。正常処理の流れ、代替処理の流れ及び例外処理の流れを別々に記述する。
18	例外処理	判定処理に於いて、「false」となったり、システムに異常が発生したりして事後条件を満たさず終了する処理の流れ。 例：パスワードを入力したが認証サーバとの通信に異常が生じ処理が abort した。
19	MVP	Minimum Viable Product の略。最小限の機能を有した製品。
20	PBI（プロダクトバックログアイテム）	プロダクトバックログの項目。

6.参考書籍

(1) 独習 UML 第4版 翔泳社 ISBN-10 4798118540

(2) アジャイルモデリング―XP と統一プロセスを補完するプラクティス 翔泳社 ISBN-10 4798102636

(3) アジャイルサムライ オーム社 ISBN-10 4274068560

(4) SCRUM BOOT CAMP THE BOOK 翔泳社 ISBN-10 4798129712

(5) スクラムガイド Scrum.org https://www.scrum.org/Scrum-Guide

(6) アジャイルな見積りと計画づくり 毎日コミュニケーションズ ISBN-10 4839924023

(7) UML モデリングのエッセンス 翔泳社 ISBN-10 4798107956

(8) Java エンタープライズ・コンポーネント―カラーUML による Java モデリング ピアソンエデュケーション ISBN-10 4894712598

(9) Software in 30 Days スクラムによるアジャイルな組織変革"成功"ガイド KADOKAWA/アスキー・メディアワークス ISBN-104048912364

(10) スクラムを活用したアジャイルなプロダクト管理―顧客に愛される製品開発 ピアソン桐原 ISBN-10 4864010978

(11) アジャイルソフトウェア開発スクラム ピアソンエデュケーション ISBN-10 4894715899

(12) 組織パターン 翔泳社 ISBN-10 4798128449

(13) XP エクストリーム・プログラミング入門(第 2 版)―変化を受け入れる ピアソンエデュケーション ISBN-10 4894716852

(14) XP エクストリーム・プログラミング適用編―ビジネスで勝つための XP ピアソンエデュケーション ISBN-10 4894715554

(15) Succeeding with Agile: Software Development Using Scrum Addison-Wesley ISBN-10 0321579364

(16) Coaching Agile Teams: A Companion for ScrumMasters, Agile Coaches, and Project Managers in Transition Addison-Wesley ISBN-10 0321637704

アジャイルソフトウェア開発向け UML 適用ガイドライン

特定非営利活動法人 UML モデリング推進協議会
アジャイルソフトウェア開発部会

ガイドライン作成メンバー

・小倉　英一郎　　ネットイヤーグループ
・大森　麻理　　東芝ソリューション株式会社
・加藤　公久　　株式会社ニコンシステム
・小阪　和宏　　東芝ソリューション株式会社
・今野　隆一　　ジャパンシステム株式会社
・正田　塁　　株式会社オージス総研
・鈴木　浩二　　スマーテックワークス株式会社
・高橋　正明　　ジャパンシステム株式会社
・竹政　昭利　　株式会社オージス総研
・田中　繁　　オープンワークス株式会社
・照井　康真　　個人会員
・長坂　健太　　個人会員
・中原　俊政　　個人会員
・仁木　卓哉　　株式会社アイ・ティ・イノベーション
・根岸　易世　　株式会社ニコンシステム
・羽生田　栄一　　株式会社豆蔵
・原田　巌　　株式会社オージス総研
・古川　剛啓　　中菱エンジニアリング株式会社
・細谷　竜一　　個人会員
・籘　耕平　　株式会社ニコンシステム
・山田　悦朗　　株式会社豆蔵


UMTP
UML Based Modeling Technologies Promotion