［論　説］

ソフトウェアの要求開発における基本概念の考察

名児耶　富美子

1．序論

製品開発の最大のリスクは、誰も欲しがらない製品を作ってしまうことである。このリスクを低く抑えるために、顧客の要求の本質を見極めて、顧客の要求を満たす品質を、定められた期間内に、適切な価格で提供することが不可欠である。近年では、システム開発時に品質に次いで、特に開発スピードが重視される［JUAS2021］。ビジネス環境の急激な変化に、迅速に対応するために、開発者はアジャイル型開発の理念や背景を理解し、実践することが推奨されている［IPA2020］［内山 1997］。アジャイル型開発手法は単一のソフトウェア開発の方法論ではなく、エクストリームプログラミング（XP）［Beck2015］、スクラム［ScrumGuide2020］、リーン［Poppendieck2003］、クリスタル［Cockburn2004］など独立した方法論や手法が提唱している要素を組み合わ

せて適用するソフトウェアの開発手法の総称である。アジャイル型開発の特徴[AgilePrinciples2001]は、開発プロジェクトの初期段階から顧客を巻き込んで製品を設計し、動く製品を短期間で実装し、顧客からのフィードバックを得ることで検証を実施し、フィードバックで学習した内容に基づいて開発者が設計を修正するといった一連の流れを短期間で繰り返すことである。

特に、アジャイル型開発の理念[AgileManifesto2001]は、開発者が「大掛かりな事前作業」によって文書を作成する仕様作成よりも、実際に動くソフトウェアを優先すると述べている。言い換えれば、アジャイル型開発手法の提唱者らにすれば、実際に開発現場で使ってきた構造化分析、オブジェクト指向、デザインパターンなどのシステム開発の上流工程での要求分析や仕様作成の作業は「大掛かりな事前作業」となる。アジャイル型開発手法の要求開発では、開発者がシステムで何を実装しなければならないかを定義する仕様書を作成する代わりに、初期段階から顧客と継続的にコミュニケーションを取ることによって、ユーザーの視点から見たソフトウェアのフィーチャーを開発者が収集する。フィーチャーは、ソフトウェアの機能、特性や特徴、性能目標、見た目や使い勝手などユーザーにとっての価値を意味する。アジャイル型開発ではステークホルダーへのインタビューやワークショップなどを実施して、フィーチャーを収集し、ユースケース、シナリオやユーザーストーリーなどの手段を使って要求を表現する[Cockburn2000]。多くのアジャイル型開発手法で推奨されているユーザーストーリーはシステムやソフトウェアのユーザーや顧客にとって価値があると思われる機能[Cohn2004]に関する要素を小さな紙のカードを使って表したものである。良いユーザーストーリーには次のような特徴があると[Rasmusson2011]は指摘する。要求が独立していてスコープを柔軟に調整でき、予算の範囲に収まるように交渉の余地があり、顧客が対価を支払っても良いと考える価値があり、見積もることができ、小さく、テストできるというという。

しかしながら、ソフトウェアが高い品質を確保するためには、要求のスコー

プを制約条件（事前条件、事後条件、不変表現）で定義することが必要であり、また、小さくするには適切な大きさのモジュールに分割する作業を実施しなくてはならない。さらに、効率的なテストを設計する上でデータの定義や制約条件が欠かせない。これら制約条件やデータの定義を集合、関係、列、関数や述語論理などの数学的な表記を使って抽象的に表す VDM［Jones1990］、Z 記法［Woodcock1996］、B メソッド［Abrial1996］、Alloy［D. Jackson2012］などの形式仕様記述を利用することにより、プログラムで実装すべき要求が明確になるだけでなく、ツールを使った検証が可能となる。ここで注意すべきことは、形式仕様記述は訓練を受けた一部の設計者には理解できる抽象的な記述であるが、一般の顧客には理解しがたい表現ということである。開発者と顧客の双方が「何を実装するのか」を合意しながら進めるアジャイル型開発に、開発プロジェクトに応じてソフトウェアの品質を確保するために必要な分だけ形式仕様記述の技術の一部を取り入れる適切な配分が必要になると考える。

本稿では、形式仕様記述の技術を開発実務に取り入れることを容易にするために提案された形式工学手法 Structured Object Oriented Formal Language (以下、SOFL)［Liu2004］の段階的仕様作成アプローチを紹介した上で、この段階的仕様作成アプローチとアジャイル型開発を組み合わせた実験を簡潔に説明し、要求開発において考慮すべき概念は何かを関連研究から考察する。さらに、ソフトウェアやシステム開発における主要な要素を抽出対象、手段、成果物の観点から抽出し、国際標準、知識体系ガイド、要求開発に関連する研究を用いて比較を行うこととする。ソフトウェアの要求開発で考慮すべき基本概念に対する考察や比較を行うことで、アジャイル型開発に形式仕様記述を一部取り入れることを将来的に可能にし、ソフトウェアの品質向上と開発現場の負担軽減に貢献するものと考える。

2．段階的仕様作成アプローチ

形式工学手法 SOFL では、最初に日常的に用いる自然言語（日本語や英語）で記述された仕様を作成し、段階的に形式仕様に変換するアプローチをとっている。この段階的仕様変換アプローチを最初に説明し、アジャイル型開発で用いられるユーザーストーリーから非形式仕様を作成して段階的に半形式仕様へ変換した実験［Nagoya2021］の概要と結果を記述する。

2.1　SOFL による段階的仕様作成アプローチ

SOFL は形式手法 VDM（Vienna Development Method）の仕様記述言語である VDM-SL（Vienna Development Method - Specification Language）［VDM-SL1996］にデータフロー図［DeMarco1979］とペトリネット［Reisig 1985］を統合した仕様書の構造を記述する手法である。図 1 は、SOFL の形式仕様が段階的に作成されるアプローチを説明している。

図 1 の左側に描かれている非形式仕様は、(1) 実装すべき機能（Functions)、

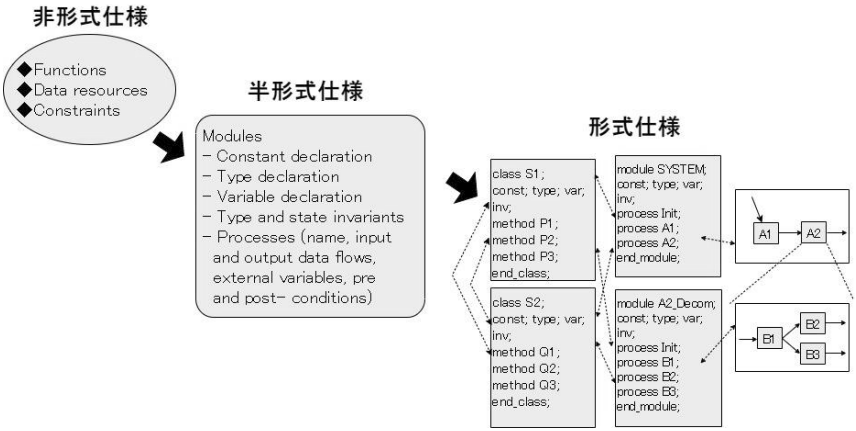

図 1：段階的仕様作成アプローチ

(2) 使用されるデータリソース (Data resources)、(3) 機能とデータリソースにおいて必要な制約条件 (Constraints) が、自然言語で明瞭かつ簡潔に記述される。図 1 の中央に描かれている半形式仕様は、非形式仕様で定義した機能、データリソース、制約条件を使って、モジュール (Module) に抽象化する。ここでいう抽象化とは、プログラムの実装に最も重要な情報を抽出することであり、それ以外の情報は捨て去ることを意味する。モジュールは定数の宣言 (Constant Declaration)、データ型の宣言 (Type Declaration)、変数の宣言 (Variable Declaration)、データ型や状態の不変条件 (Type and State Invariants)、事前条件や事後条件などから構成されるプロセス (Process) が含まれる。プロセスでは、プロセスの名前、使用されるインプットデータフローとアウトプットデータフロー、外部のデータストアとデータストアへの操作が定義されるが、事前条件や事後条件は自然言語で記述されるのが半形式仕様の特徴である。図 1 の右側に描かれた形式仕様は、階層化されたデータフロー図において生じるプロセスと、モジュール、そのモジュールに関連するクラス (class) の関係が VDM-SL の形式仕様記述を用いて定義され、構造化分析 [DeMarco1979] とオブジェクト指向 [Meyer1997] を取り入れた構造になっている。

2.2 仕様の段階的詳細化の実験

段階的仕様作成アプローチとアジャイル型開発を組み合わせた実験で行った概要を説明する。ビジネスを専攻する大学 3−4 年生が SOFL とアジャイル型開発を 6 か月間学んだ上で、モバイルアプリケーションの開発を担当し、12 のプロジェクトを実施した。対象とするビジネス領域については、知識や関心がある分野、またはアルバイトやインターンシップ等の労働経験がある分野を選択した。ベビーシッターとシッターを雇いたい親のマッチング、アルバイトのシフト管理、傘のレンタル、地下道のバリアフリー案内などを目的としたスマートフォン向けアプリケーション等の要求開発をフィードバックによる修正

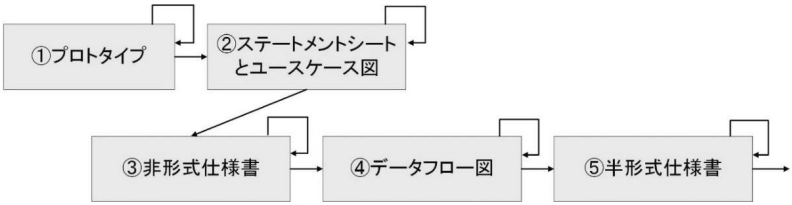

図 2：反復型開発による仕様詳細化の実験

を短期間で繰り返すイテレーション（以下、反復型開発）で実施した。プロジェクトは、プロジェクトの責任者 1 名、チームメンバー 4 名〜8 名、司会進行 1 名で構成され、図 2 にある成果物が作成された。

各プロジェクトの責任者は、アプリケーションが対象としているビジネスモデルの分析仮説を行った後に、図 2 の①プロトタイプと②ステートメントシートとユースケース図を作成しユーザーストーリーを記述した。その上で、ユーザーストーリーから③SOFL の非形式仕様、④データフロー図、⑤SOFL の半形式仕様を段階的に作成した。ただし、数学的な表記を使って抽象的に表す記述を一部のチームメンバーしか理解できないことから、事前条件や事後条件を自然言語で記述する半形式仕様までを作成することとし、形式仕様は作成しなかった。

それぞれの工程で 2 週間に一度フィードバックに基づいて成果物の修正が実施された。このフィードバックと成果物の修正記録から修正時点や修正内容を抽出し、修正内容をインターフェース、システムの領域、機能・サービス、入出力データ、制約条件に分類した。その結果、システムの領域は前半の工程（図 2 の①や②）だけでなく後半の工程（図 2 の③〜⑤）でもフィードバックで修正されていることが判明した。さらに、システムの領域に関する分析が足らないために非形式仕様以降の工程で手戻りが発生することが分かった。

前の工程に戻ってやり直すことによって生じるスケジュールの遅延を防ぐために要求開発で何をするべきかが問題になる。そこで、まず、要求開発で考慮

すべき基本概念を検討することから始めたい。

3．要求開発において考慮すべき概念

ソフトウェアの要求開発とは、要求抽出、要求分析、仕様作成、妥当性確認を含む概念である。ここでは、要求開発に関わる概念を整理する。前述のように段階的仕様作成アプローチとアジャイル型開発を組み合わせた実験では非形式仕様以降の工程で手戻りが発生していた。このことは、ユーザーストーリーで要求分析が足りていないために、非形式仕様の記述が十分ではなかったことを意味している。ユーザーストーリーは「誰のために、何を達成したいのか、そして、それはなぜか」を記述しているユーザーや顧客の視点から見たソフトウェアの機能に関する要素［Rasmusson2011］である。一方、SOFL の非形式仕様は、機能（Functions)、データリソース（Data resources)、制約(Constraints）をシステムの設計者の視点から定義している。つまり、ユーザーストーリーから非形式仕様を作成するには、ユーザーや顧客の視点からシステムの設計者の視点へ変更が必要となる。ところが、先に述べた段階的仕様作成アプローチとアジャイル型開発を組み合わせた実験では、ユーザーストーリーから非形式仕様を作成する段階でユーザー自身の振る舞いとシステムの振る舞いの混合が複数のプロジェクトにおいて発生していた。この混合が後の工程においてフィードバックで指摘され、手戻り作業を行う原因となっていた。ユーザーの振る舞いは人間の動作であってシステムの外に存在する要素であり、実装すべきシステム内部の特性ではない。言い換えれば、ユーザー自身の振る舞いはシステムが動作する世界を構成する人間の要素に関する記述であって、実装するシステムの機能やサービスに関する記述ではないが、こうしたシステムの外に存在する要素の振る舞いはユーザーストーリーには含まれていると考える。

3.1　ドメインとマシンの概念

アジャイル型開発手法と従来の要求工学に基づく要求開発プロセスの違いについて比較した Bertrand Meyer は、ドメインとマシンの概念に対する考慮がアジャイル型開発には足らないと説明している［Meyer2014］。ドメインとマシンの概念とは、Pamela Zave と Michael Jackson によって説明されている「ドメインと呼ばれるシステムが動作する世界（W）の下で、システムの振る舞い（B）はマシン（M）によって作られるのであり、システムが動作する世界（W）とマシン（M）を混合してはならない」という主張［Zave1997］［M. Jackson2000］である。

動物園の入場ゲートシステムを対象システムとして例に挙げ、システムが動作する世界（W）は、動物園の入園者、コインを入金する装置、入園者を通すように回転するバーで構成されているとして、［M. Jackson2015］では以下のように説明する。ステークホルダーが望む「入場ゲートシステムではコインの入金がない人間は入場させない」という要求は、システムの振る舞い（B）として定義される。システムの振る舞い（B）は、プログラミングによって作られるマシン（M）の振る舞いと、システムが動作する世界（W）にある動物園の入園者、コインを入金する装置、そして回転バーの振る舞いによって構成される。プロジェクトで設計する必要があるのは、システムが動作する世界（W）の振る舞いではなく、マシン（M）の振る舞いであるとしている。

3.2　対象システムの要求と環境

システム開発の現場において、対象システムを取り巻く環境が複雑すぎて、対象システムと環境を明確に区別できない事態が要求で発生する。システムを取り巻く環境には、人々、組織、規則、デバイス、影響する物質的なシステム、その他の関連するシステムなどが含まれる。形式手法の研究者らはシステムを取り巻く複雑さに対応するために数学的なモデルを用いて、要求を抽象化することで高い信頼性が確保する方法を長年の研究で提案してきた［中島 2012］。

開発現場では開発の期間や予算が限られているため、要求を抽象化する時間を確保することは非常に難しい。品質の確保と開発スピードが同時に求められるアジャイル型開発の現場で、対象システムの要求と環境の混合を防ぐためには、現実的にどのように対応すれば良いのだろうか。

システムの設計者の視点で対象システムの要求と環境から何を抽出し、分析し、仕様書で定義すべきなのかを、要求開発の用語を定義した標準 ISO/IEC/IEEE24765［IEEE2017］、知識体系ガイド SWEBOK［SWEBOK2014］、Karl Wiegers らによる研究［Wiegers2013］を比較することによって明らかにしたい。

ISO/IEC/IEEE24765 の正式名称は ISO/IEC/IEEE 24765: 2017 Systems and software engineering - Vocabulary で、システムとソフトウェア工学に関する共通の語彙の提供を目的とした国際標準である。SWEBOK は、Guide to the Software Engineering Body of Knowledge を SWEBOK と称し、IEEE Computer Society が後援して編集されたソフトウェア知識体系ガイドである。［Wiegers2013］は Karl Wiegers と Joy Beatty による著書『Software Requirements』で、その中でソフトウェアの要求開発を引き出し、分析、仕様作成、妥当性確認の分野に分けて、概念を構成する要素を丁寧に説明していることから本研究の趣旨に一致するものと考えた。

原文からの翻訳は著者が直接行い、コメントで著者の見解を述べている。比較対象に一致する用語がない場合には、著者がもっとも近い表現であると考える用語を選んで比較を行うとする。

4．基本概念の比較

最初に［IEEE2017］、［SWEBOK2014］、［Wiegers2013］を使って、ソフトウェアやシステム開発の対象としての要求の定義から始め、アジャイル型開発が収集しているフィーチャー、仕様書で記述される制約条件を比較する。次に

アジャイル型開発で要求を表現するのに利用される手段としてのユースケース、シナリオ、ユーザーストーリーを検討する。最後に要求が記述される成果物としてのシステム要求、機能要求、非機能要求について比較を行う。それぞれの基本概念について共通している点や特徴的なことをコメントで述べることとする。最後に、コメント全体について考察を行う。

4.1　要求、フィーチャー、制約条件

要求とは具体的に何を指しているのか、フィーチャー（feature）として何を収集すべきか、仕様書で記述される制約条件とは何を指しているか、ソフトウェアやシステム開発における抽出対象を比較する。

4.1.1　要求（requirement）

- 要求［IEEE2017］

  ①ニーズとそれに関連する制約と条件を翻訳または表現したもの。②同意、標準、仕様、またはその他の正式に課された文書を満足するようなシステム、システムの構成要素、製品、またはサービスが満たす、あるいは所有する必要がある条件または能力。③満たすべき基準を内在する規定。④契約またはその他の正式に課された仕様を満たす製品、サービス、または結果に存在しなければならない条件または能力。

- ソフトウェアの要求［SWEBOK2014］

  現実世界における問題を解決するために提示すべきプロパティ。

- 要求［Wiegers2013］

  顧客のニーズや目標、またはそのようなニーズや目標を満たすために製品が持たなければならない条件や能力を記述したもの。ステークホルダーに価値を提供するために製品が持つべきプロパティ。

コメント：［SWEBOK2014］では、要求という用語の直接の定義はなく、ソ

フトウェアの要求という用語で記載されている。ソフトウェアの要求［SWEBOK2014］と要求［Wiegers2013］で述べられているプロパティとは、真か偽かを判断できるブール型の述語で表現された特徴である。つまり、要求にはブール型の述語が記述されることになる。

4.1.2 フィーチャー（feature）

- フィーチャー［IEEE2017］
  ①システムアイテムの際立った特性、②機能的または非機能的な際立った特性、多くの場合は既存のシステムの強化、③エンドユーザーやその他のステークホルダーが理解できる対象システムの抽象的な機能特性。
- フィーチャー［SWEBOK2014］
  フィーチャーの記述は見つからなかった。
- フィーチャー［Wiegers2013］
  ユーザーに価値を提供し、かつ、機能要求の集合として記述される、論理的に関連する１つ以上のシステムの能力。

コメント：フィーチャー［IEEE2017］③とフィーチャー［Wiegers2013］から、フィーチャーは具体例ではなく抽象度が高い記述であると考えられる。

4.1.3 制約条件

- 制約条件［IEEE2017］
  ①システム工学の工程で設計ソリューションまたは実装を制約し、かつ、企業によって変更できない規制または暗黙の要求。②ソフトウェア・ライフサイクルプロセス（SLCP）の開発上の制約。③データの有効な条件を指定するルール。④制約条件が満たされるために真であることが要求される事実の陳述であることの責任。⑤１つ以上の価値や存在に基づくオブジェクトの属性の値、または、存在に対する制約。⑥システム要件、設計、実装上、また

は、開発プロセスやシステム変更において、外部から課せられた制限。⑦プロジェクト、プログラム、ポートフォリオ、またはプロセスの実行に影響を与えるような制限を加える要因。

- 制約条件［SWEBOK2014］
  解決方法によって満たすべき制約、または、実行可能な解決方法を制限する可能性がある制約。
- 制約条件［Wiegers2013］
  設計と製品の構築のために開発者が利用可能な自由選択肢に課される制限。他の種類の制約条件には、プロジェクトマネージャが利用可能な選択肢に制限を課すものもある。ビジネス・ルールは、ビジネスの運用上、ソフトウェアシステムの理由上ゆえに、制約条件をたびたび課すことがある。

コメント：制約条件［IEEE2017］⑥にある外部から課せられた制限や、制約条件［Wiegers2013］の後段の記述にあるビジネスの運用上の変更などの理由で制約条件が後から追加されることが著者が実施した仕様の段階的詳細化の実験［Nagoya2021］でも多く発生した。

4.2　ユーザーストーリー、ユースケース、シナリオ

アジャイル型開発で要求を表現する手段として一般に利用されているユーザーストーリー、ユースケース、シナリオについて比較を行う。

4.2.1　ユーザーストーリー

- ユーザーストーリー［IEEE2017］
  ①ソフトウェアの機能が満たすユーザーの目標を簡単な誰にでも伝わるような文章で描かれた記述。②ソフトウェアシステムとの望ましいユーザーインタラクションの説明として表示されるソフトウェアの要件、機能、フィーチャー、または品質属性についての誰にでも伝わる記述。

●ユーザーストーリー［SWEBOK2014］

顧客の用いる言葉で表現された、求められている機能についての短い概要。典型的なユーザーストーリーの形式は「誰が〈役割〉、何を望むのか〈ゴール/望み〉、何のために〈便益〉」となっている。ユーザーストーリーは、開発者が製品を実装するための努力の合理的な見積もりを作成できるように、ちょうど十分な情報を含むことを目的としている。

●ユーザーストーリー［Wiegers2013］

アジャイル型開発のプロジェクトにおいて 1〜2 行程度の文章で記述されるユーザー要求の獲得するための書式で、ユーザーのニーズを明確にしたり、望ましい機能の単位を記述したりすると同時に、ユーザーにとっての機能の便益を述べている。

コメント：ユーザーストーリー［IEEE2017］やユーザーストーリー［SWEBOK2014］では、ナラティブ（narrative）、つまり、開発者が話し手となり、話し手側の頭の中で理解した内容を一連の出来事として顧客の使う言葉で表現するという態度が重視されていると考える。

4.2.2 ユースケース

●ユースケース［IEEE2017］

① UML において、アクターに測定可能な価値の結果を提供するシステムの全部のタスク。②システムがユーザーと対話し、ユーザーに測定可能な価値の結果を提供することで実行可能な一連のタスク。③システムの振る舞いに関する要求とユーザーとの相互作用に関する記述。

●［SWEBOK2014］

ユースケースの記述は見つからなかった。

●ユースケース［Wiegers2013］

アクターとシステムの間の一連の論理的に関連する可能性のある相互作用に

関する記述で、これにより、アクターに価値を提供する結果に至る。複数のシナリオを含むことが可能。

コメント：UML［UML2017］のユースケース図におけるユーザーとシステム、または、アクターとシステムとの相互作用に関する記述がユースケース［IEEE2017］とユースケース［Wiegers2013］で共通して述べられている。

4.2.3　シナリオ

- シナリオ［IEEE2017］
  脚本の意味。同時、または連続して発生する一連のイベントについてステップごとの記述。
- シナリオ［SWEBOK2014］
  シナリオは、ユーザー要求を引き出すためのコンテキストを提供する価値ある手段を提供する。シナリオは、「もしも～したら」と「どのように行われるか」という質問を許可することで、ユーザータスクに関する質問のフレームワークをソフトウェアエンジニアに提供する。最も一般的なタイプのシナリオは、ユースケースの記述である。
- シナリオ［Wiegers2013］
  あるゴールを達成するための、ユーザーとシステム間の特定の相互作用に関する記述。または、システムで使用されるインスタンス、またはユースケース間の特定のパス。

コメント：アジャイル型開発で要求を獲得するためのシナリオは、シナリオ［IEEE2017］で述べている一連のイベントについてステップごとの記述と考える。一方、シナリオ［SWEBOK2014］とシナリオ［Wiegers2013］の後半で記述されているシナリオは、UML［UML2017］のユースケース図におけるユースケースの記述に関することが定義されている。ユースケースの記述では、ユー

スケースを実行したときの基本フロー、例外フローなどの処理の流れをステップごとに文章で表したものがシナリオと呼ばれる。

4.3　システム要求、機能要求、非機能要求

システムで何を実装するか、どのような振る舞いを提供するか、どのような特性や属性を持っているのかを記述した成果物がなければ、現実的にはシステム開発の契約はできない。システム開発実務では要件定義という工程で作成する成果物になるが、ここでは原文の Requirement を「要求」として翻訳し、システム要求、機能要求、非機能要求と表現し、それぞれが、どのように定義されているかを確認する。

4.3.1　システム要求

- システム要求仕様［IEEE2017］

  システム要求仕様は、①システムの要求を具体化する情報で構造化された集まり、②システム、そのシステムの運用環境および外部インターフェースに関する要求（機能、パフォーマンス、設計上の制約条件、および属性）が構造化された集まり。

- システム要求［SWEBOK2014］

  システム要求はシステム全体のための要求である。ソフトウェアコンポーネントを含むシステムにおいては、ソフトウェア要求はシステムの要求から導き出される。この知識体系では、システムの顧客またはエンドユーザーの要求として、ユーザー要求は制限する方法で定義する。対照的に、システム要求では、ユーザー要求、他のステークホルダー（規制当局など）の要件、および個人が識別可能な情報源以外の要求が含まれる。

- システム要求［Wiegers2013］

  すべてソフトウェア だけ、または ソフトウェア と ハードウェア から構成されると思われる複数のサブシステムを含む製品の大まかな概要。

コメント：[IEEE2017] では、システム要求という用語の直接の定義はなく、システム要求仕様という用語で記載されている。ここではシステム要求よりも、仕様が「構造化された集まり」を定義しているとの説明が中心になっている。システム要求 [SWEBOK2014] は、システム要求に関わる対象をシステムの顧客またはエンドユーザーに限定せず、ステークホルダーを含み、さらに個人が識別可能な情報源以外の幅広い人的情報源を対象にしていることは注目に値する。システム要求 [Wiegers2013] はシステムの内部構造を中心に説明している。

4.3.2 機能要求

- 機能要求 [IEEE2017]
  ①製品またはプロセスが生み出す結果を特定する陳述。②システムまたはシステムコンポーネントが実行する機能を指定する要求。
- 機能要件 [SWEBOK2014]
  機能要件は、たとえばテキストの書式を整えることや信号を変えることなど、ソフトウェアが実行するべき機能を述べている。それらは、性能またはフィーチャーと呼ばれることもある。機能要件は、その動作を検証するために書かれたテストステップの有限集合としても記述できる。
- 機能要件 [Wiegers2013]
  ソフトウェアシステムが特定の条件下で発揮すると思われる振る舞いの記述。

コメント：機能要件 [SWEBOK2014] では、具体的な機能を例に挙げるとともに、別の呼び方やテストとの関連を記述していて、最も詳細な定義になっている。機能要件 [Wiegers2013] では、「特定の条件下で発揮すると思われる」という表現で、範囲を限定した定義になっている。

4.3.3 非機能要求

- 非機能要求［IEEE2017］
  性能属性。ソフトウェアが何をするかではなく、ソフトウェアがどのようにそれを行うかを記述するソフトウェア要求。
- 非機能要件［SWEBOK2014］
  非機能要求は、解決策を制約するために働く要求である。非機能要件は、制約または品質要求としても知られている。それらは、パフォーマンス要求、保守性要求、安全性要求、信頼性要求、セキュリティ要求、相互運用性要求、または他の多くの種類のソフトウェア要求の１つであるかどうかにより、さらに分類することができる。
- 非機能要件［Wiegers2013］
  システムが示さなければならないプロパティまたは特性に関する記述、またはシステムが尊重しなければならない制約条件に関する記述。

コメント：非機能要求については、ソフトウェアの性能や品質、システムが持つべきプロパティまたは特性という点にそれぞれ言及している。その一方で、非機能要件［SWEBOK2014］と非機能要件［Wiegers2013］の両者で制約を共通して挙げている。

4.4 考察

要求においてプロパティ（ブール型の述語）の記述が、フィーチャーにおいて具体例ではなく抽象度が高い記述が要請されていた。制約条件では外部から課せられた制限だけでなく、ビジネスの運用上の変更で後から追加されることが明確になった。

ユーザーストーリーは開発者の頭の中で理解した内容を顧客の使う言葉で表現するナラティブな態度が求められ、顧客重視のアジャイル型開発手法の思想が反映されていた。その一方で、ユースケース、シナリオについては UML の

概念が中心に定義されていたため、アジャイル型開発で対象システムの要求と環境からユースケース、シナリオを使って、何を抽出すべきなのかは判断できなかった。

システム要求、機能要求、非機能要求では定義がそれぞれに異なっていたが、特に注目した点は知識体系ガイドにおけるシステム要求の定義である。システム要求に関わる対象をシステムの顧客またはエンドユーザーに限定せず、ステークホルダーを含み、さらに個人が識別可能な情報源以外の幅広い人的情報源にまで及んでいた。

5．結論

この論文では、形式工学手法 SOFL を用いた段階的仕様作成アプローチを説明し、段階的仕様作成アプローチとアジャイル型開発を組み合わせた実験の結果を踏まえ、要求開発において考慮すべき概念の考察を行った。まず、アジャイル型開発で用いられるユーザーストーリーから SOFL による段階的仕様作成アプローチの最初の成果物である非形式仕様を作成するには、ユーザーや顧客の視点からシステムの設計者の視点へ変更が必要であるが、このユーザーストーリーから非形式仕様への変換時に、開発対象システムを取り巻く環境が複雑すぎて対象システムの要求と環境の混合が発生していることが明確になった。次に、対象システムの要求と環境の混合を防ぐために、システムの設計者の視点で対象システムの要求と環境から何を抽出すべきなのかを要求開発の用語を定義した国際標準、知識体系ガイド、関連研究から比較を行った。その結果、要求、フィーチャー、制約条件では抽象度が高い記述が求められていることが判明した。さらに、システム要求においては、システム要求に関わる対象がステークホルダーや個人が識別可能な情報源以外の幅広い人的情報源まで範囲に含まれていることが分かった。

今後は、システムのステークホルダー別や人的情報源による要求分析に関す

る関連研究を精査し、仕様の段階的詳細化にステークホルダー別や人的情報源別で対象システムを分析する方法を検討し、要求の妥当性検証に利用したいと考える。

参考文献

[Abrial1996] J.-R. Abrial: The B-book: assigning programs to meanings. Cambridge University Press, USA, 1996.
[AgileManifesto2001] アジャイルソフトウェア開発宣言
https://agilemanifesto.org/iso/ja/manifesto.html
[AgilePrinciples2001] アジャイル宣言の背後にある原則
https://agilemanifesto.org/iso/ja/principles.html
[Beck2015] ケント ベック、シンシア アンドレス：『エクストリーミングプログラミング』、オーム社、2015.
[Cockburn2004] Alistair Cockburn: Crystal clear a human-powered methodology for small teams (First. ed.). Addison-Wesley Professional, 2004.
[Cockburn2000] Alistair Cockburn: Writing Effective Use Cases (1st. ed.). Addison-Wesley Longman, 2000.
[Cohn2004] Mike Cohn: User Stories Applied: For Agile Software Development. Addison Wesley Longman, 2004.
[DeMarco1979] Tom DeMarco: Structured Analysis and System Specification. Prentice Hall PTR, 1979.
[D. Jackson2012] Daniel Jackson: Software Abstractions: Logic, Language, and Analysis. The MIT Press, 2012.
[IPA2020] アジャイル開発版「情報システム・モデル取引・契約書」
https://www.ipa.go.jp/ikc/reports/20200331_1.html
[IEEE2017] ISO/IEC/IEEE 24765:2017: Systems and software engineering - Vocabulary.
https://www.iso.org/standard/71952.html
[Jones1990] Cliff B. Jones: Systematic software development using VDM (2nd ed.). Prentice-Hall, Inc., 1990.
[JUAS2021] 日本情報システム・ユーザー協会「企業 IT 動向調査報告書 2021」
https://juas.or.jp/cms/media/2021/04/JUAS_IT2021.pdf
[Liu2004] Shaoying Liu: Formal Engineering for Industrial Software Development. SpringerVerlag, 2004.
[Meyer1997] Bertrand Meyer: Object-oriented software construction (2nd ed.). Prentice-Hall, Inc., 1997.
[Meyer2014] Bertrand Meyer: Agile!: The Good, the Hype and the Ugly. Springer, 2014

[M. Jackson2000] Michael Jackson: Problem frames: analyzing and structuring software development problems. Addison-Wesley Longman Publishing Co., Inc., 2000.
[M. Jackson2015] Michael Jackson: Behaviours as design components of cyber-physical systems. In: Meyer B., Nordio M. (eds) Software Engineering. LASER 2013, LASER 2014. Lecture Notes in Computer Science, vol 8987. pp.43-62. Springer, Cham. 2015
[中島 2012] 中島 震：『形式手法入門―ロジックによるソフトウェア設計―』、オーム社、2012.
[Nagoya2021] Fumiko Nagoya: A Case Study on Combining Agile Requirements Development and SOFL. In: Jinyun Xue, Fumiko Nagoya, Shaoying Liu, Zhenhua Duan (eds.) Structured Object-Oriented Formal Language and Method - 10th International Workshop, SOFL+MSVL 2020, Singapore, March 1, 2021, Revised Selected Papers. LNCS, vol.12723, pp. 23-33. Springer 2021
[Poppendieck2003] メアリー・ポッペンディーク、トム・ポッペンディーク、平鍋健児他(翻訳)：『リーンソフトウエア開発～アジャイル開発を実践する 22 の方法～』、日経 BP、2004.
[Rasmusson2011] Jonathan Rasmusson, 西村直人他 (監訳)：『アジャイルサムライ―達人開発者への道』、オーム社、2011.
[Reisig1985] Wolfgang Reisig: Petri nets: an introduction. Springer-Verlag, 1985.
[ScrumGuide2020] The Scrum Guide,
https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100
[SWEBOK2014] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society: Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0 (3rd. ed.). IEEE Computer Society Press, 2014.
[UML2017] OMG: Unified Modeling Language v2.5, 2017.
https://www.omg.org/spec/UML/2.5.1/About-UML/
[内山 1997] 内山悟志：『デジタル時代のイノベーション戦略』、技術評論社、2019.
[VDM-SL1996] ISO/IEC 13817-1:1996: Information technology - Programming languages, their environments and system software interfaces - Vienna Development Method - Specification Language - Part 1: Base language,
https://www.iso.org/standard/22988.html
[Wiegers2013] Karl E Wiegers and Joy Beatty: Software Requirements 3. Microsoft Press, 2013.
[Woodcock1996] Jim Woodcock and Jim Davies: Using Z: specification, refinement, and proof. Prentice-Hall, 1996.
[Zave1997] Pamela Zave, Michael Jackson: Four dark corners of requirements engineering. ACM Transactions on Software Engineering and Methodology, Volume 6, pp. 1-30, 1997.