

要件定義にまつわる問題と その解決に向けて

ver. 1.0

ValueSource

▶▶▶ <http://www.vsa.co.jp>

technologies
from SAPPORO



要件定義にまつわる問題と その解決に向けて

この資料の内容

ソフトウェア開発プロセスと要件定義

この資料で取り扱う要件定義の、ソフトウェア開発全体における位置付けと範囲について説明します。

要件定義の目標

この資料が扱う主題である要件定義の目指すべき完成像について説明します。

要件定義にまつわる問題

要件定義の作業を進める現場で多く見られる事例を紹介しながら、そこにある問題と原因を考察し、前のセクションで説明した要件定義の完成像との関係を説明します。

要件定義に求められるもの

要件定義に不可欠な、表現力、網羅性、整合性、基本コンセプトのそれぞれについて、その理由や背景について説明します。

リレーションシップ駆動要件分析(RDRA)

以上の課題の解決や目標を達成する上で有効な要件分析手法として、当社が提唱している「リレーションシップ駆動要件分析」の概要を説明します。

まとめ

資料全体の結論をまとめます。

この資料の内容	1
ソフトウェア開発プロセスと要件定義	1
要件定義の目標	1
要件定義にまつわる問題	1
要件定義に求められるもの	1
リレーションシップ駆動要件分析 (RDRA)	1
まとめ	1
1. ソフトウェア開発プロセスと要件定義	3
2. 要件定義の目標	4
3. 要件定義にまつわる問題	5
3-1. 要件定義で多く見られる事象	5
3-1-1. 全体像が見えてこない	5
3-1-2. 顧客との打合せをリードできない	7
3-1-3. まとまらない会議	8
3-1-4. シーズ中心	9
3-1-5. 議論がすぐに袋小路に入る	10
3-2. 問題と原因の整理	11
4. 要件定義に求められるもの	13
4-1. 表現力と UML	13
4-1-1. 複雑な要求の分析	14
4-1-2. 顧客との打合せ	14
4-1-3. 合意形成	14
4-1-4. 基本コンセプトの検討	14
4-2. 網羅性	15
4-3. 整合性	15
4-4. 基本コンセプト	15
5. リレーションシップ駆動要件分析 (RDRA)	17
5-1. 要件定義をうまく進めるには	17
5-2. 基本ダイアグラム	18
5-2-1. システム価値	19
5-2-2. システム外部環境	19
5-2-3. システム境界	20
5-2-4. システム	21
5-3. 関係ダイアグラム	22
5-3-1. システム外部環境の関係ダイアグラム	22
5-3-2. システム境界の関係ダイアグラム	23
5-3-3. システムに関する関係ダイアグラム	23
5-4. トレーサビリティチェック	24
まとめ	25

1. ソフトウェア開発プロセスと要件定義

ソフトウェア開発という活動は、一般に右図のような作業プロセスで構成されます。

開発するソフトウェアには、顧客またはエンドユーザーが抱えている問題を解決したり、あるいは必要としている何らかの機能を満たすことが求められます。

この目的を果たすために、まず対象となる業務を分析して、ITを有効に活用したビジネスの手法や価値を明確にする必要があります。これがビジネスモデリングです。

業務の内容が把握できたら、今度はその業務に現状どのような問題があり、どのように解決することが望まれているのかを洗い出し、さらに、そのためにソフトウェアがどのような機能を提供する必要があるのか、ということ定義する必要があります。また、システムとして問題をどのように実現するのかというアーキテクチャの観点から出される要求もあります。こうして出された要求は、最終的にソフトウェアの仕様に反映されることとなります。

このように、ソフトウェア仕様の原点となる要求事項を洗い出し、整理していく活動が**要件定義**です。**要件定義**は、完成したソフトウェアの価値を左右する非常に重要なプロセスであるといえます。

この資料では、ソフトウェア開発におけるこの**要件定義**の部分を扱います。要件定義において一般によく見られる問題点とその原因を考察を示し、問題を解決する一つの答えとして当社が提唱しているリレーションシップ駆動要件分析(RDRA)の手法について説明します。

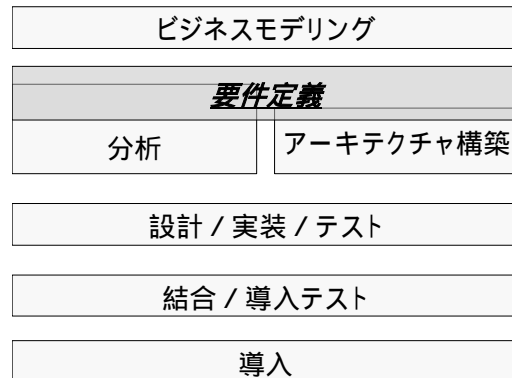


図 1: 一般的な開発工程

2. 要件定義の目標

要件定義に取り組むにあたっては、まず、その目標を理解すると共に、最終成果物のイメージを把握しておくことが重要です。

要件定義とは、その言葉どおり、システムに必要な条件を定義することです。しかし、個々の条件を列挙することが要件定義の本質ではありません。本当に重要なことは、これから作っていくシステムの基本コンセプトを語ることにあります。個々の要件は、そのコンセプトを具現化するための手段とそこで必要となる条件を表したもので、いわばコンセプトを表現するための手段と捉えることもできます。要件を一つ一つ定義していくことによって、システムの価値の根源となるコンセプトの輪郭を浮かび上がらせるわけです。個々の要件を洗い出していく中で、システムの基本コンセプトを明確に打ち出すこと、それが要件定義の本質的かつ最も重要な目標です。

ただし、要件を洗い出して列挙するだけで自動的に基本コンセプトが明確になるわけではありません。むしろ反対に、そうした目標を達成できるように要件の洗い出しを行うことが重要です。

明確なコンセプトに基づいて各要件が適切に洗い出された要件定義は、次の3つの特性を兼ね備えていると考えます。

- * 網羅性: コンセプトの具現に必要な要件がもれ、重複なく示されている
- * 整合性: 各要件が基本コンセプトおよび他の要件と整合したものになっている
- * 表現力: それぞれの要件がわかりやすく表現されている

こうした特性を持つ成果物によってシステムの明確なコンセプトが語られること、あるいはそのような成果物を作っていく過程でコンセプトを打ち出していくことが、要件定義という活動の目標です。

上の説明だけでは、これら3つの特性がなぜ重要なのがよく理解できないかもしれません。そこで次のセクションでは、要件定義において多く見られる事例を見ながら、これら3つの特性が重要な理由について説明していくことにします。

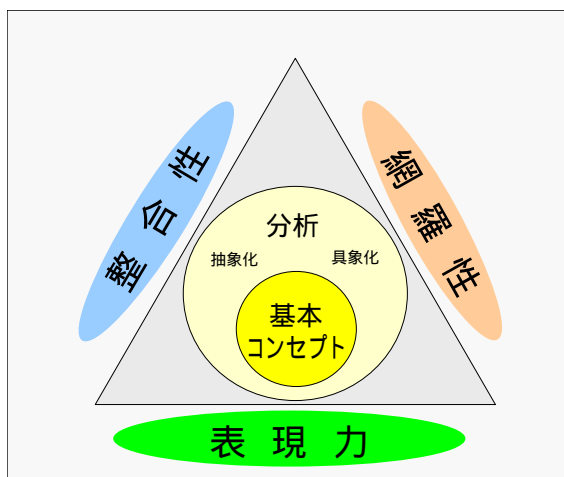


図 2: 要件定義に求められるもの

3. 要件定義にまつわる問題

3-1. 要件定義で多く見られる事象

前述のように、要件定義においては、まずシステムの明確な基本コンセプトがあること、加えてそれを網羅性、整合性、表現力という3つの特性を兼ね備えた成果物として表現することが重要です。これらのいずれが欠けていても、要件定義の活動で問題が発生することになります。以下、こうした事例を見ながら、上記の要素が重要な理由を考えてみましょう。

3-1-1. 全体像が見えてこない

次の図は、新しいソフトウェアの開発、既存製品への新機能の追加などを目的として立ち上げられたプロジェクトで多く見られる状況を表しています。

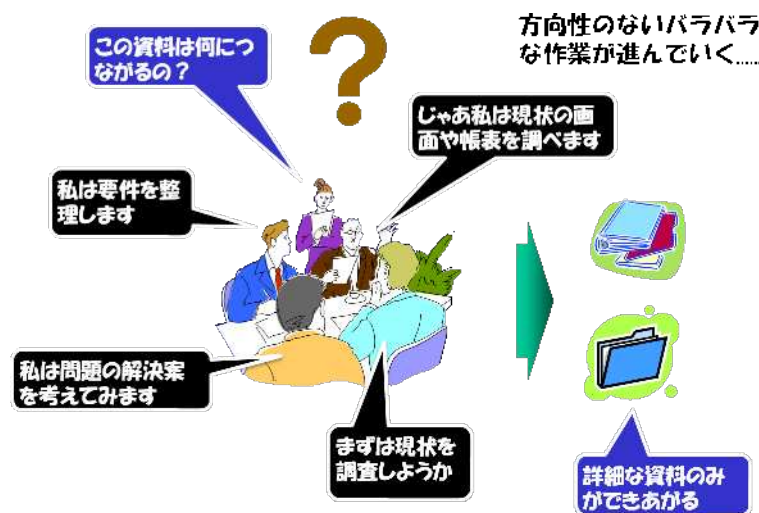


図 3: 方向性のない作業

要求の整理、既存システムの調査、問題解決策の検討など、一見、彼らは妥当な作業をしているように見えます。各担当者の作業成果が提出され、プロジェクトで作成したドキュメントの量も順調に増えていきます。成果物ができ上がっていく様子を見たマネージャも、プロジェクトは問題なく進行していると考えているようです。

こうした作業を一ヶ月も続けると、膨大な量のドキュメントが完成します。しかし、いくらドキュメントを作っても、いっこうにシステムの全体像が見えてきません。やがてプロジェクトのメンバーは、「こんな資料を作って一体何の役に立つのだろうか?」という疑問を抱き始めるようになります。このように、実態としては、着手できる作業をとりあえず先に進めているだけなのに、あたかも要件定義をしているような「つもり」になっているプロジェクトを多く見かけます。

この問題の原因は、要件定義として最終的に完成させる成果物の全体像が見えていないこと、その目標に向けて要件をまとめていく思考の枠組みがないところにあります。

要件定義では、全体の整合性を保ちながら相互に関連しあう要件を網羅的に表現する必要があります。ここに適切な仕事の進め方のヒントがあります。つまり、網羅性によって必要な情報の枠組みが決まり、整合性を求めることから作業手順が見えてくることになるのです。網羅性、整合性を意識した成果物の目標を持たずに、機械的に担当を割り振って個人レベルの

作業に突入してしまうと、この事例のように、出てくる成果物は方向性のないものとなり、それらを集めてみてもシステムの全体像は浮かび上がってこないのです。

3-1-2. 顧客との打合せをリードできない

ソフトウェア開発の上流工程では、関係者との打合せを通じて要望や意見を収集および分析し、加えて様々なアイデアを出しながら、要件定義の完成という目標に導いていかなければなりません。最初から答えが自明であるような場合を除き、単に人間が集まって意見を出しあうだけで方向性が自然に定まることはありませんから、関係者が納得するシステムの全体像をイメージした上で、戦略的に打合せをリードしていくことが重要になります。ただし、それは必ずしも容易なことではありません。

開発者側で打合せをリードすることの難しさは今に始まったことではありませんが、近年その度合いがますます高まっているように思われます。ひとつの背景としては、開発者側にスキルと経験が不足していること、具体的には、プロジェクトを最初から最後まで経験したことのない技術者に要件定義が任されるケースが多くなっていることがあります。たとえば、システムの保守を中心に仕事をしてきた人が年齢的な理由でプロジェクトリーダーとなり、そのまま要件定義を担うようになった事例や、ユーザ企業のシステム部門が要件をとりまとめる事例、あるいは、大手 Sier の担当でシステム構築の実経験のない人間が要件定義を行っている事例などが挙げられます。

このような例では、経験不足の担当者が要求をどう組み立てれば良いのかが分からず、教科書的な手順（最初に業務フローを書き、次に XXX を作成し、・・・）に基づいてを漫然と作業を進めている状況が多く見られます。

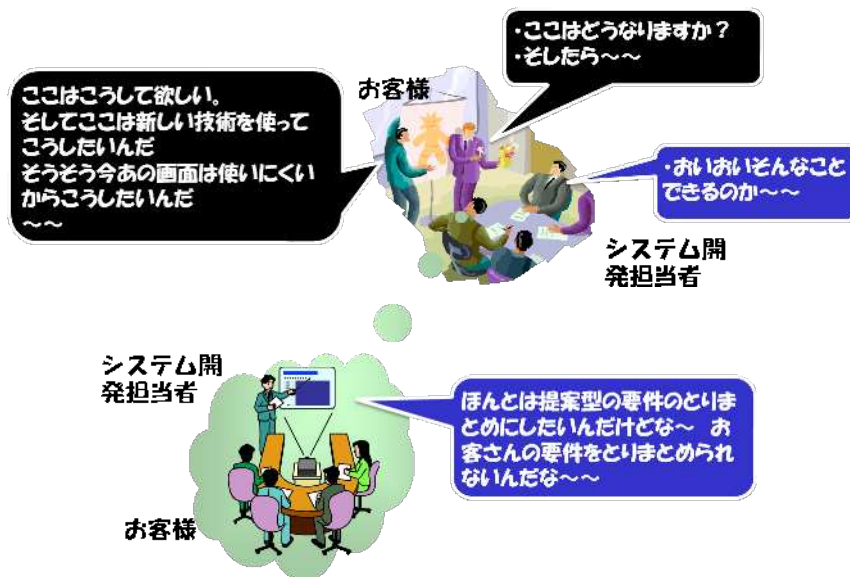


図 4. 顧客をリードできない会議

このようなやり方ではユーザに説得力のある提案ができず、会議のイニシアチブを握ることも難しくなります。プロジェクトを混乱させないためにも、総合的に見て妥当と思われる提案をあらかじめ用意し、会議の場で積極的に示していくことで、議論を適切な方向に導いていく必要があります。そのためには、システム開発全般についての知識と構想力、アイデアを積極的に提示していく行動力、そして考えを人に伝える表現力が必要です。

開発者側主導で要件定義を提案し、それを関係者が納得して受け入れる状況を作るには、その内容がわかりやすい形で表現されていることが大前提となります。関係者が納得できるように提案を検討し、UML などの図でわかりやすく表現した資料を用意しておくことで、提案ベースで会議をリードしやすくなり、そうすることで、顧客の多様な要望による会議の混乱を防ぐことにつながります。高い表現力によって内容の理解が深まれば、参加者の思考が活性化され、より発展的に議論を展開させることも可能になります。要件定義の資料にはその元となる表現力が必要なのです。

3-1-3. まとまらない会議

下記は、ある開発会議でのやりとりです。

1月10日

Aさん「私は新しい製品に Web サービスの機能を入れるのが良いと思う。なぜなら・・・」
Bさん「これからはリッチクライアントが主流になると思う、そうすることで Web 系のシステムと違いが出せる。」
Cさん「私は現在の製品に XXX の問題があるから、それを解決するために YYY の機能を入れるのが良いと思う。」

今日のところは最初の会議でもあり各自の意見(イメージ)を言って終わる。

1月13日

Aさん「最近読んだ本におもしろい記事があって、我々が開発するシステムもその技術を採用するのがいいと思う。」
Bさん「大手ベンダーの XXX が最近 SHT を提唱している。今後はそれが主流になるから我々もそれでいこう。」
Cさん「営業が今の製品で苦勞している部分から DDD を改善した機能があるのがいいな。」

2月10日

Bさん「やはり製品開発にはビジョンが大切だから今日はビジョンを考えてきたよ。プロジェクトがスタートして1ヶ月経過したけど、そろそろ基本的な機能だけでもまとめた方がいいけど・・・。」

上記の開発会議の流れを見ていると、以下のような傾向が読み取れます。

- * Aさんは新しい技術に興味がある
- * Bさんは新しい製品は既存のものにない新規性を強く打ち出したいと考えている
- * Cさん現状の問題やその改善に関心が向いている

これは、実際よく見かける風景です。メンバーの意識も高そうですし、具体的に作業を始めています。しかし、しばらくすると、会議に参加しているメンバーは一ヶ月前から議論が何も前進していないと感じ始めます。最初は高かった意識も徐々に低下していきます。

やりとりを見直してみると、毎回の会議で、全体の合意が何も取り交わされていないことがわかります。各人がただ言いたいことを言って終わっているようにも見えます。本来、会議の目的はメンバー間でアイデアを共有し、その上で合意を形成していくことですが、それを行わないまま形式的にいくら会議を繰り返しても、議論が収束することはありません。

会議を実りあるものにするには、何を合意するのかという目的を明確にすることと、その合意を得るための手段を持つことが重要です。前者のためには、要件定義として決定すべき事項が網羅的にリストアップされていることが前提となります。後者については、合意の出発点としてアイデアを共有することが必要ですから、やはりここでも、表現力のある形で考えを資料にまとめて人に伝えることが重要になります。

3-1-4. シーズ中心

システムについてのアイデアがあり、ブレインストーミングによって有望なアイデアがいくつもできたと思います。そのアイデアを検証または分析すべく、役割を決めて作業を始めます。そして、その結果を持ち寄り、またブレインストーミングします。プロトタイプを作成し、実現可能性や機能の有効性を検証します。一ヶ月がたち、アイデアがそこそこ形になってきます。Ajax、SOA、DI など、この業界流行りの魅力的なセールストークがちりばめられています。

でも、リーダーは何故か「これでいいのだろうか」という疑問を感じています。「分かってきたことは沢山あるけれども、その一つ一つがどうもうまくつながっていかない。細部を見ると面白いことや今までになかったものができ上がってくるように思える...でも結局いまひとつ製品の全体像がつかめていない。このやり方を続けていって本当にシステムのイメージが固まるのだろうか...」。

このような状況を経験したことはないでしょうか。

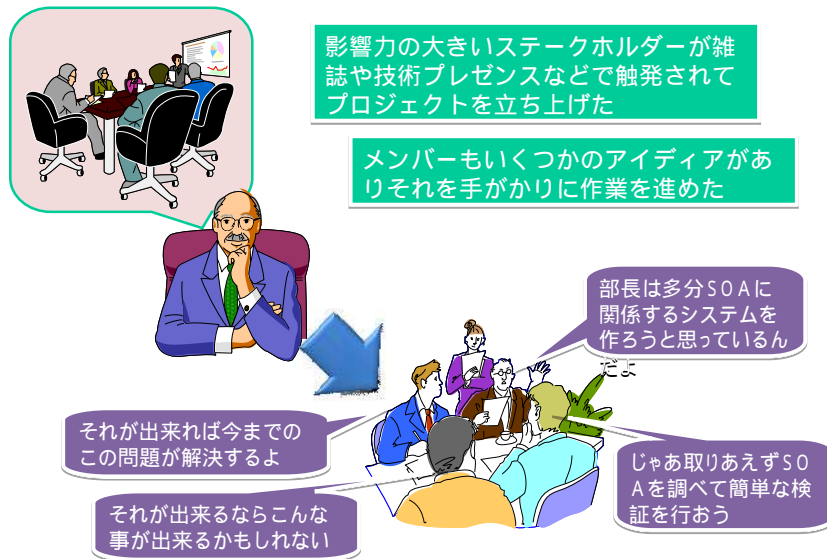


図 5: 利用者の姿が見えていない会議

アイデアを出し、その実現可能性を検証し、その有効性を把握するというのは、漠然としたものを具体化するにはごく当たり前のやり方です。しかし、実際に製品開発をやったことがある方なら経験があると思いますが、この繰り返しだけでは製品の全体像がつかめてこないのです。漠然としたニーズをベースにその時々シーズを組み立ててみてもバラバラな要素の集まりにしかなりません。

この時しっかりとした製品コンセプトがあり、その実現可能性が見えている場合は、製品の方向性を見失うことはないのですが、コンセプトそのものがぐらついている時は上記のように試行錯誤を繰り返すこととなります。つまり、そのシステムはいったい誰がどのように嬉しいのかということを明確にイメージできる製品コンセプトを打ち出す必要があるのです。

製品コンセプトが必ずしも最初から見えているとは限りません。むしろ、要件定義の過程でコンセプトが固まっていくということも多いでしょう。要求の本質が何なのかを考えることによって、そのシステムが提供すべき価値の本質が見えてくるわけです。そのためには、個々の具体的な要求を抽象的に表現するための手段を持つことが重要になります。抽象化によって本質的な構造が洗い出されれば、それを基に製品のコンセプトにまで高めることができます。

3-1-5. 議論がすぐに袋小路に入る

打合せを行っていて議論が袋小路に入ってしまうことがあります。問題に対して打つ手がなくなると議論がストップするか、話がループしてしまいます。大抵は「次回までに各人で解決案を考えてくるように…」ということになって、そこで打ち合わせが終わります。要件定義では特に、要求事項とその実現可能性の問題で議論が止まってしまうがちです。

以下、コンサルティングの中で経験した事例から、要件定義の場面における要求の衝突として代表的なものを挙げます。

競争力のある要求 実現可能性が高いものを実現

製品開発においては競争力のある仕様を実現しようとする、一般的に過去に経験の無いことが要求されます。その結果開発スキルの問題や納期、マンパワーなどの問題により、一気に実現可能性が低くなる場合があります。

マーケットへの投入時期 開発期間は十分に欲しい

マーケティングの見地からいくとその製品の投入時期というのはとても大事な事です。しかし、開発現場に銀の弾丸は存在せず、コツコツと開発を進める必要があります。この二つを両立することはとても難しい問題です。

仕様は一度に決められない 仕様を決めないと開発に入れない

現在のソフトウェアは様々な事ができるようになっています。その結果あることを実現するにも何通りもの実現方式があり、それぞれに一長一短があります。それらの組み合わせが製品の価値を決めます。そしてそれを紙の上だけでイメージして仕様としてまとめるのはとても難しいことです。一方ものの作りは作るもののはっきりしないと作れないのも事実です。

- ・ 枝葉にこだわると議論が先にすすまない
- ・ 現状の問題にばかり目が向いている
- ・ 現状の制約を前提条件としてしまう



図 6: 議論が進展しない会議

複雑な要求が絡み合って解決の糸口が見出せなくなった場合は、要求を構造化して整理し、構造間の関係をひもといっていくことで問題解決の糸口を探ります。

また、競合する要求によって思考がストップしてしまう場合は、問題の本質がどこにあるのかをもう一度考え直す必要があります。競合が起きている範囲だけに思考がとどまっている限り、がんじがらめの状況から抜け出すことはできないからです。

こうした作業を行っていく上でも、要件定義を視覚化して表現する能力が重要になります。各要求間の関係を整理してその構造をわかりやすく表現したり、具体的な要求を抽象化することで、より全体的な視点から問題の解決方法を探ることができます。

3-2. 問題と原因の整理

これまで見てきた事例の問題点を振り返って整理してみると、次のようにまとめることができます。

問題点	原因
全体像が見えてこない	要件定義として最終的に完成させる成果物の全体像が見えない。 その目標に向けて要件をまとめていく思考の枠組みがない。
顧客との打合せをリードできない	開発者側にスキルと経験が足りない。 脈絡のない話をまとめる手段がない。 考えを整理して人に伝えるための表現手段がない。
まとまらない会議	要件定義において何を合意するべきなのかが明らかでない。 合意の前提として必要な、アイデアを表現して共有する手段がない。
シーズ中心	システムの明確な製品コンセプトがない。 コンセプトを固めていくために必要な表現手段がない。
議論が袋小路に入る	細部にこだわり、全体的な視点から問題解決を探ることがない。 複雑に絡み合った要求をひもとくための構造化や抽象化の表現手段がない。

表 1

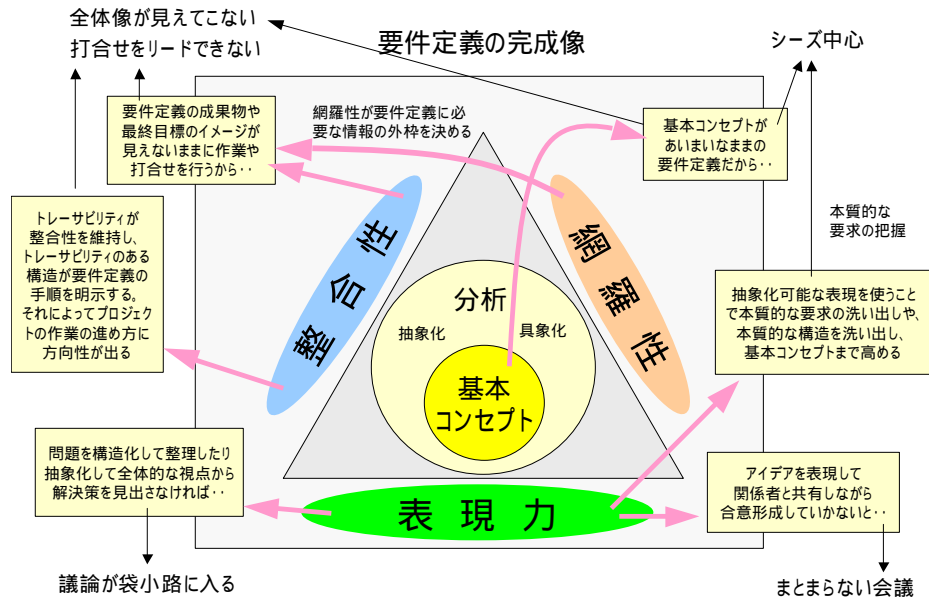


図 7: 事例と要件定義の関係

各事例の解説の中でも触れたように、これらの問題はすべて、要件定義に必要な基本コンセプトと3つの特性のいずれか欠けていることに由来しています。事例で示した問題点と要件定義の完成像との関係を図で表すと次のようになります。

以上、5つの事例とその考察を通じて、要件定義では、網羅性、整合性、表現力を兼ね備えた成果物でシステムの基本コンセプトを語ることが重要であることを説明しました。

次に、そうした成果物を実際に作成する作業について検討していきます。

4. 要件定義に求められるもの

ここでは、要件定義に求められるものと、それを得るための方法について説明します。

4-1. 表現力と UML

先の事例の考察で表現力の重要性を繰り返し指摘しました。顧客との打合せ、合意形成、基本コンセプトの検討、要求の分析など、要件定義という活動のさまざまな局面で、メンバーの思考を共有し、拡張していくためには、入り組んだ情報を整理し、わかりやすい図的な表現方法で視覚化するのが有効です。

このための手段として UML のモデルを利用できます。UML には、異なる複数の視点(ビュー)からシステムを捉えるダイアグラムが用意されており、この図を適切に用いることでシステムの全体像を表現できます。

UML (Unified Modeling Language) は、オブジェクト指向分析や設計モデリングで共通して使用される標準的な表記法です。現在広く使用されている UML 2.0 では、次のように分類される 13 種類のダイアグラムが用意されています。

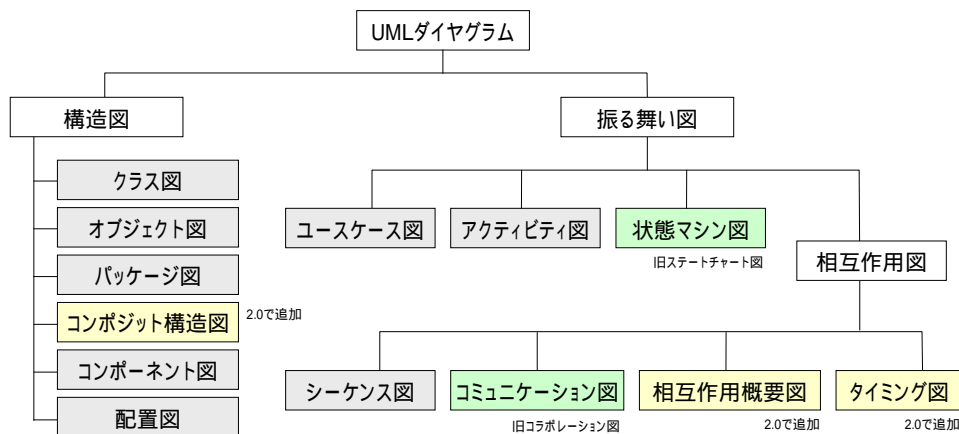


図 8: UML ダイアグラムの分類

構造図	クラス図	クラスの仕様やクラス間の関係などを定義
	オブジェクト図	インスタンスとその関係を定義する
	パッケージ図	パッケージ(モデルの要素を分類する)を定義する
	コンポジット構造図	クラスの内部構造を定義する(UML 2.0 で追加)
	コンポーネント図	コンポーネントを定義する
	配置図	システムの物理的な構成を定義する
振る舞い図	ユースケース図	システムが提供する機能を定義する
	アクティビティ図	アクションの実行順序を定義する
	状態マシン図	状態と状態遷移を定義する
	シーケンス図	クラス間の相互作用を時系列に定義する
	コミュニケーション図	クラス間の相互作用をクラス間の関係に着目して定義する
	相互作用概要図	相互作用図の実行順序を定義する
	タイミング図	相互作用と状態遷移に関する時間制約を定義する

表 2

なお、この資料では UML そのものの詳細については扱いません。必要に応じて専門の本や資料などを参照してください。

UML の表現力は、要件定義のさまざまな局面で役立てることができます。以下、いくつかの例を挙げます。

4-1-1. 複雑な要求の分析

先の事例でも触れたように、システムの全体像が見えないという問題の原因は、要件定義を行う思考の枠組みがないことですから、その枠組みとなるものを用意すれば問題を解決できます。UML は、この枠組みの中で複雑な要求を整理する手段として利用できます。対象を抽象化または具象化したモデルを UML のダイアグラムとして表現できます。

また、図を使って手早くアイデアを表現できれば、会議の場で資料を作成し、そのままメンバー全員で共有することができます。UML のモデルは、その手段としても利用できます。資料作成は個人作業とみなされがちですが、全体の方向性が定まるまでは会議の中で共同して進める方がうまくいきます。まずは、見えているところから作業に着手し、会議の場で方向性を確認しながら、その場で完成できない詳細部分などを個人作業として割り当てるようにします。

このように、プロジェクトの各メンバーが協調しながら一つの方向に向かって作業を進めるようになると要件が固まり始め、システムの全体像が見えるようになります。

4-1-2. 顧客との打合せ

顧客との打合せは話が脱線することも多く、それらとも整合性をとりながら打合せの内容を組み立てていくには、システムの根幹部分と枝葉の部分を明確に押さえておく必要があります。図でわかりやすく表現した資料を用意しておくことで、顧客の多様な要望や関心で会議を混乱させることが少なくなります。また、要件定義までの道筋が見えていることで、打合せのペース配分とその打合せで決めておくべき事が明確になり、提案ベースで戦略的に会議をリードできます。

4-1-3. 合意形成

合意プロセスにおいても対象の可視化(見える化)が有効です。一般に、合意のプロセスでは、次の点に注意する必要があります。

* **何を合意するのか？**

対象を可視化し、それを共有することで合意対象を明らかにします。

* **どのように合意するのか？**

会議の場において、資料を作りながら、その場で合意します。

* **合意したものをどのように表現するのか？**

ポンチ絵や特定の表記規則に基づく図的な表現形態を利用します。

この可視化手段として UML を利用できます。合意の対象となるものを UML のダイアグラムで可視化することで、より効率的に合意が行えるようになります。

4-1-4. 基本コンセプトの検討

システムのコンセプトを検討するには、システム化する対象を多角的に捉える必要があります。システムの機能や利用方法を多角的に記述していくと、それらを集めた全体を通して、システムが生み出す価値の本質が見えるようになってきます。さらに、個々の要求を抽象化していったときに、そのシステムの本質的な価値が何なのかを簡潔に語る事ができれば、基本コンセプトが見出せたこととなります。

こうした検討においても、イメージがわかりやすい図的なモデルの利用が効果的です。UMLには異なるビューに対応するモデルのダイアグラムが用意されているので、それらを利用してさまざまな角度からシステムを記述できるほか、抽象化や具象化の概念も表現できるので、個別の要求から本質的なコンセプトを見出す作業に効果的です。また、この場合も、アイデアを視覚化できる成果物を共同で作りながら議論を組み立てていく方法が有効です。この成果物が以降の議論のベースとなり、それを共通の基盤としてさらにアイデアを発展させていくことができます。

4-2. 網羅性

一般に、システムには複数の側面があり、対応する視点(ビュー)から性質を記述することで、全体像を表現することができます。逆に、各ビューからシステムの性質が漏れなく記述されているかどうかをチェックすることで、網羅性の確保につなげることができます。UMLはこのビューの考え方が基本になっており、各ビューに対応するものとしてダイアグラムと表記法が定義されています。要件定義においても、これら複数のビューを意識して対象をモデル化することで、網羅性を確保することができます。

また、システム内からシステム外へとつなぐ考え方も網羅性に有効です。具体的には、要件定義を次のような4つの領域として対象を捉え、業務とシステムをつなげていきます。

1. システムの価値を捉える
2. システムの外部環境を捉える
3. システム境界を捉える
4. システムの内部構造(データ、ドメイン、ビジネスルール)を捉える

この枠組みに沿って分析作業を進めていくことで、要件定義の網羅性を確保できます。

4-3. 整合性

ビューに基づいて洗い出した要件は通常、単独で存在するわけではなく、システムの他の部分や別の要求と関連しあって存在しています。したがって、要件定義においては、個々の要件を洗い出すだけでは不十分で、それらが全体として整合性のあるものになっていることを確認する必要があります。

このためには、常に全体と部分の両方の視点を持ち、モデル同士を関係づけながら要件をブレイクダウンしていくことが重要です。具体的には、異なるビューに基づく複数の基本モデルを1つのダイアグラム上に描き、お互いの関係をチェックしながら作業を進める方法が有効です。こうした手法は、個人作業だけでなく、打合せの議論にも適用することもできます。

個々のビューに対応するモデルを表現した図(基本ダイアグラム)のセットによって網羅性を保証したうえで、さらに複数の基本モデルを1つの図上に表現した図(関係ダイアグラム)を利用して整合性を検証することができます。

4-4. 基本コンセプト

新しいシステムを作り出す時はそのシステムを関係者もしくはユーザに説明しそれが理解されることが重要です。機能のてんこ盛りを説明されても、システムの価値を感じることはありません。システムを人に説明する場合、そのシステムを使うとどのような事ができて、それがどんなに嬉しいのか、役に立つことなのか、また楽しいことなのかを説明するでしょう。開発するシステムについても同じことが言えます。システムが持つ本質的な価値として語れる基本コンセプトの存在が重要なのです。その上で各々の機能がどのように関わりを持つのか明らかになってくると、自信を持って「このシステムはこうです」と語れるようになります。

現実のプロジェクトにおいて、製品コンセプトが最初からはっきりしていることはまれで、ましてそのコンセプトが適切なものであるという例はさらに少ないでしょう。多くの場合、製品コンセプトは要件定義をしていく中でメンバーの頭の中に芽生えてくるものです。それが何なのかという答えをここで明らかにすることはできません。なぜなら、その答えを生み出すのは、みなさん自身だからです。

ただし、そのための活動を効果的に進めるための方法なら示すことができます。次のセクションでは、この資料の締めくくりとして、当社が提唱するリレーションシップ駆動要件分析 (RDRA: Relationship Driven Requirement Analysis) という要件分析手法の概要について説明します。

5. リレーションシップ駆動要件分析 (RDRA)

5-1. 要件定義をうまく進めるには

まず、要件定義という作業を上手く進めるための条件について考えてみましょう。

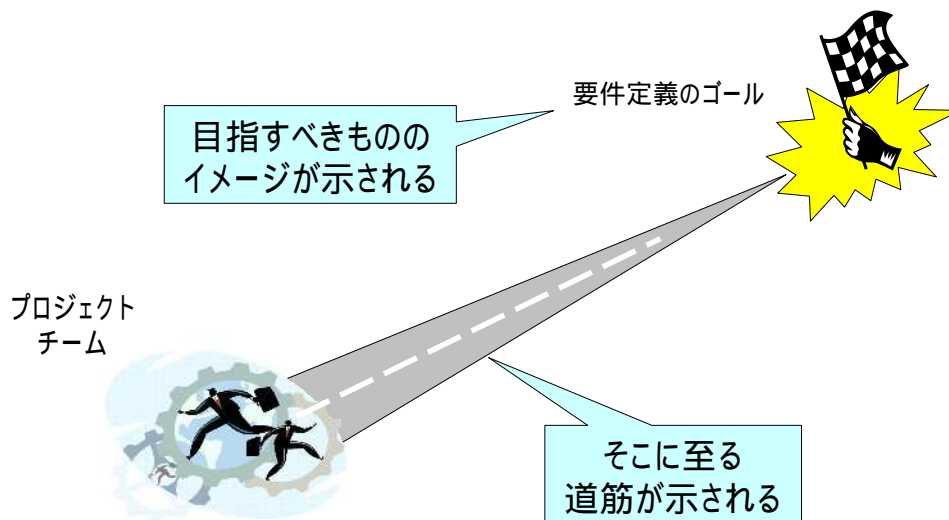


図 9: 要件定義の道筋

ひとつは、目指すべきもののイメージを明確にしておくことです。的確に表現された要件定義のイメージを念頭に置いて作業に取り組むことは、常にシステムの本質的な価値を考えながら要件を洗練化させていくことにつながります。これには成果物の表現形態をあらかじめ明確にしておくことが有効です。具体的には、UML の表現力を利用して網羅的かつ整合性のある形で要件を表現するダイヤグラムのセットを定義しておくことです。

もう一つは、それらのダイヤグラムを使って、どのように作業を進めていくのかという手法を明らかにしておくことです。言い換えれば、最終成果物という目標に至るまでの道筋を示すということです。

当社では、これらの観点から、UML モデリングに基づいて要件定義を効果的に進めるための手法として、リレーションシップ駆動要件分析 (RDRA: Relationship Driven Requirement Analysis) を提唱しています。RDRA では、成果物を規定する UML ベースの表記法とダイヤグラムのセット、およびそれを作成するための手法という大きく 2 つの要素で構成されています (なお、この手法ではダイヤグラムに UML の表記法を利用しますが、必ずしも厳密な UML の表記規則に縛られるものではありません)。

以下、この手法の概要を紹介します。

5-2. 基本ダイアグラム

RDRA では、システムの要件を UML のモデルを利用して複数の視点から表現します。既に述べたとおり、要件定義に UML のモデルを適用することで、アイデアを視覚化してプロジェクトメンバーと共有したり、抽象的な思考を支援してコンセプトの導出に役立てるなど、要件定義のさまざまな場面で効果を上げることができます。

RDRA では、大きく次の 4 つの領域に分けて要件を記述します。

- * システム価値(コンテキスト、要求モデル)
システムは誰にとってどのような価値があるのかを要求として明らかにする
- * システムの外部環境(業務モデル、利用シーンモデル、概念モデル)
システムの価値を実現するための業務や利用シーンを明らかにし、システム境界を決める前提を明確にする
- * システム境界
(ユースケースモデル、画面・帳票モデル、イベントモデル、プロトコルモデル)
ユーザとシステムとの境界を明示することでシステムとして必要なインターフェースを明らかにする
- * システム(データ/ドメイン、機能、ビジネスルール)
システム境界を実現するための機能とデータを明らかにする

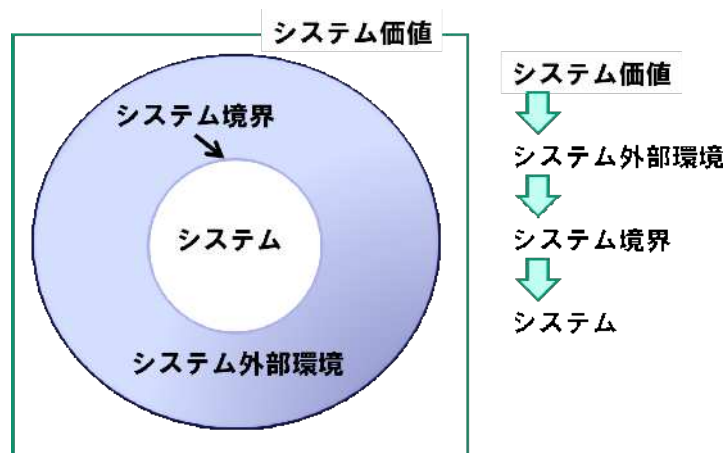


図 10: 要件定義の 4 つの領域

以下、それぞれの分類別に構成概念と対応するダイアグラムを概説します。

5-2-1. システム価値

「システム価値」では、業務に関わる人、システムに関わる人から要求を集め洗練化しながら要件へと整理します。関係者の要求をヒアリングして構造化し、その中で粒度を調整しながら本質的な要求、そして要件へと洗練化していきます。

ここに表現するモデルでシステムの価値を決める人(ロール)とその価値を定義します。

コンテキストモデル

まず、システムへの要求を発生させる元になる利害関係者を洗い出すためにコンテキスト図を描きます。コンテキスト図は、**開発時コンテキスト**と**運用時コンテキスト**に分けて作成します。

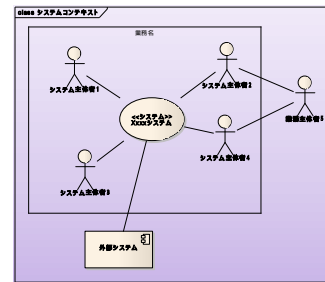


図 11

要求モデル

要求は**機能要求**と**非機能要求**に分けられます。UMLで定義されている表記法ではありませんが、右図のような要求アイコンを使用して要求モデルのダイアグラムを作成します。

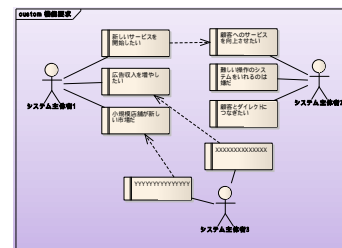


図 12

5-2-2. システム外部環境

「システム外部環境」では、業務フローまたは利用シーンを使用して、先にあげたシステム価値を表現します。同時に、業務の中で認識されている概念を明らかにします。これにより、システムの境界線を決める前提となる外部環境の認識を関係者の間で合わせます。

業務フロー

コンテキスト図に現れた人(組織)がどのように連携しながら業務を進めていくのかを明らかにします。

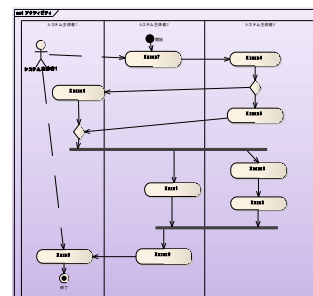


図 13

利用シーン

システムがどのように利用されるのかが具体的にイメージできるように、物語風の説明を個々の要求に対応づけて記述します。

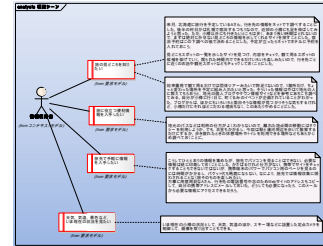


図 14

概念モデル

業務の中で認識されている概念を表す用語の意味とそれらの間の関係をモデル化してクラス図で表し、プロジェクトのメンバー間で認識を合わせます。

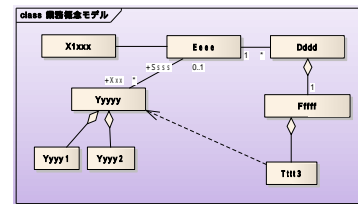


図 15

5-2-3. システム境界

「システム境界」では、システムと外部環境との境界に位置する部分を捉え、それを表現します。まず、人が関わる部分はユースケースと画面・帳表モデルでシステム境界を明らかにします。外部システムに関わる部分はイベントモデルとプロトコルモデルでシステム境界を表します。

ユースケースモデル

外部から見たシステムの機能をユースケースとして洗い出します。全てのユースケースを洗い出すことで人(組織)が関わるシステム境界を明確にしたことになります。

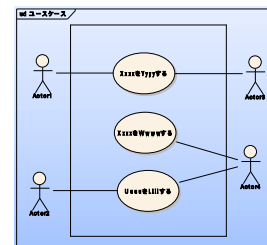


図 16

画面・帳表モデル

ユーザーとの直接のインターフェイス部分として、画面と帳表の構成および項目を明らかにします。

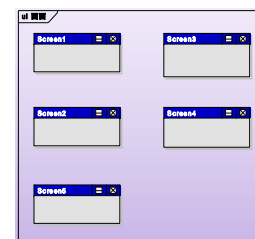


図 17

プロトコルモデル

外部システムとの連携に関する状態遷移を表現した状態マシン図です。このダイアグラムを描くことで、整合性を確保しながら次の外部イベントを洗い出すことができます。

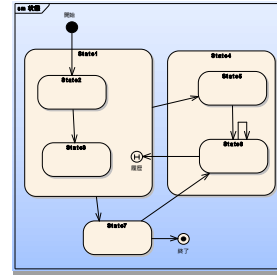


図 18

イベントモデル

プロトコルモデルから抽出され、外部システムとのインターフェイス仕様を与えます。

なお、外部システムが既存のシステムとして存在する場合は、インターフェイスとしてイベントが先に決まっていることもあります。その場合は、このシステムに関するイベントだけを取り上げ、その関係をプロトコルモデルとして明確にします。

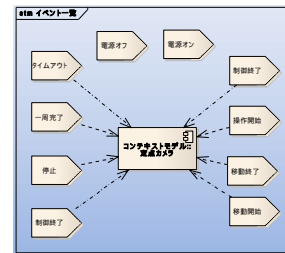


図 19

5-2-4. システム

「システム」では、システム内部構造に関する要件として、データモデルまたはドメインモデル、機能モデル、ビジネスルールを明らかにします。

データモデル/ドメインモデル

永続化を前提としたデータ構造はデータモデルとして表現します。永続化に限らず、より一般的にシステム内部のソフトウェア構造を表現する場合はドメインモデルと呼ばれます。いずれも、UML のクラスで表現します。

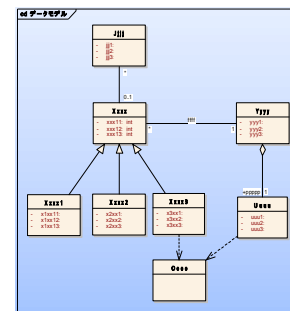


図 20

機能モデル

ユースケースやイベントを実現するために必要な機能を洗い出します。機能は複数のユースケースやイベントから利用できます。



図 21

ビジネスルール

業務上のルールとして状態を認識できる場合は、状態マシン図として表現します。

「システム境界」でプロトコルモデルとして状態マシン図を描きましたが、それとは異なり、ビジネスルールを表現するためのダイアグラムです。

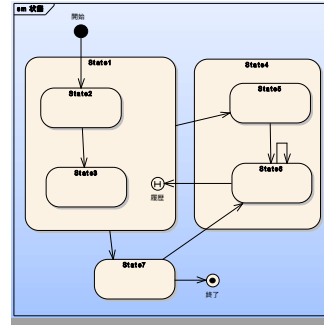


図 22

5-3. 関係ダイアグラム

RDRA では、要件定義の整合性を確保するため、先に紹介した基本ダイアグラムに加えて、関係ダイアグラムという図を使用します。関係ダイアグラムとは、各ビューの間で要件に不整合が生じていないことを確認するため、基本ダイアグラムをのモデル要素を1つの図に混在させて描いたものです。モデル間の関係を関係ダイアグラムとして描くことで、要件の整合性を確認しながら作業を進めることができます。対応する相手がいない要素が見つかった場合に、その原因を検討することによって、漏れの発見やモデルの洗練化に役立てることができます。

以下、主な関係ダイアグラムを示します。

5-3-1. システム外部環境の関係ダイアグラム

業務フロー&ユースケース

業務フローのアクティビティ図にユースケースをマッピングすることで、個々のアクティビティとユースケースとの対応関係を確認します。またレーン(パーティション)には、アクターを対応付けます。

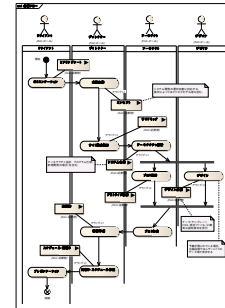


図 23

利用シーン&ユースケース

業務フローが描けない場合は、利用シーンとユースケースとの対応関係を図に表して同様の関係付けを行うことができます。

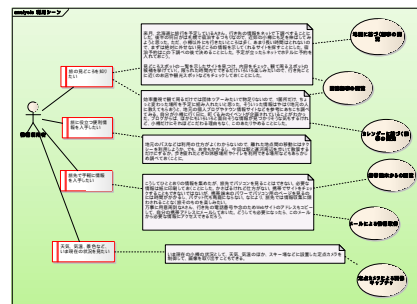


図 24

5-3-2. システム境界の関係ダイアグラム

ユースケース&画面・帳表

ユースケース図で、各ユースケースに対応する画面や帳表をマッピングします。こうすることで、どのユースケースでどの画面や帳表が使用されるのかが明らかとなり、対応関係の不整合や漏れなどの発見につながります。

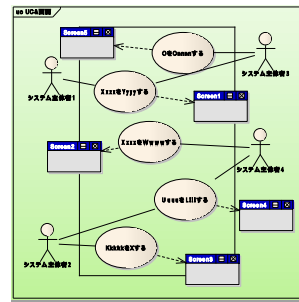


図 25

5-3-3. システムに関する関係ダイアグラム

機能複合モデル

機能を中心に、画面、ユースケース、データ、ドメインを結びつけ、全体としての整合性をチェックします。

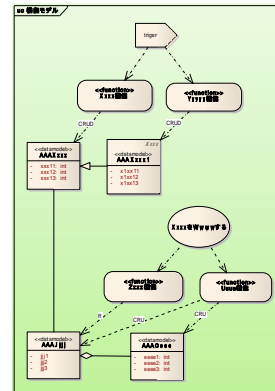


図 26

5-4. トレーサビリティチェック

上記のような関係ダイアグラムを利用することで、システムの各部分が別の部分に正しく対応していること(トレーサビリティ)のチェックに役立てることができます。基本モデル間のトレーサビリティのチェックポイントを図にまとめると次のようになります。

図 27 に書かれているノートと吹き出しが書くダイアグラム間の関係性について着目するポイントを示しています。

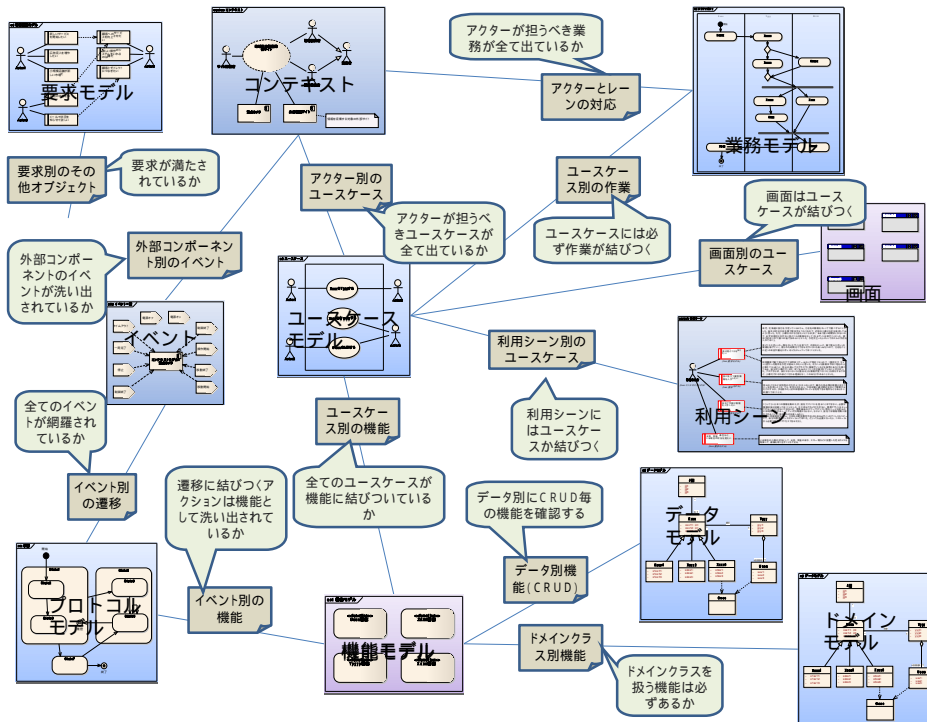


図 27: ダイアグラム間の関係

まとめ

この資料では、当社が提唱するリレーションシップ駆動要件分析(RDRA)の概要について説明しました。

最初に、ソフトウェア開発全体における要件定義の位置づけを確認した後、要件定義にまつわる問題事例とその原因を考察しました。その解決方法としてUMLを利用したモデリングが有効であることを示し、当社が提唱するリレーションシップ駆動要件分析の概要として、構成概念と基本ダイアグラム、関係ダイアグラム、およびそれらを利用したトレーサビリティチェックと全体の要件定義プロセスを簡単に説明しました。

リレーションシップ駆動要件分析の全体像をまとめると次のようになります。

1. 要件定義の枠組みの提供
 - システム価値
 - システム外部環境
 - システム境界
 - システム
2. モデルとして整理する
 - システムの価値を明らかにする
 - コンテキストモデル、要求モデル
 - システムの外部環境を把握する
 - 業務モデル、利用シーン、概念モデル
 - システム境界を明らかにする
 - ユースケースモデル、画面・帳表、イベントモデル、プロトコルモデル
 - システムの内部構造を明らかにする
 - データモデル/ドメインモデル、機能モデル、ビジネスルール
3. 分析する
 - 視点を定める What と How の分離
 - 連続的に進める トレーサビリティを利用した分析
4. トレーサビリティを保ちながら洗練化する
 - 各ダイアグラムは相互に関係を持つ
 - その関係に着目し各モデルを洗練化する
 - 洗練化の過程を通じて現状を深く知る
 - システムの本質を捉える
 - システム要件の定義に結びつける

以上のように、RDRAでは個別の視点から要件を記述するための基本ダイアグラムのセットと、それらの間の関係性を定義します。そして、その関係性を利用して関連するダイアグラムをスピーディーに作成し、かつそれらのダイアグラム間の整合性をチェックしながら洗練を進める手法です。

この手法で最も重要かつ特徴的なポイントが、モデル間の関係性を定義することです。このモデリング作業を通じて全体的な視点から要件定義を繰り返し見直すことにより、個々の視点から見た要求の羅列ではなく、それらが相互に関連しあい、かつ全体として整合性が取れた有機的なシステムの全体像を記述するものとモデルを洗練させていくことができるのです。

要件定義モデリングの具体的な手順については、モデリング編で取り扱います。

なお、この資料では扱っていない、具体的なモデリング作業やモデルの洗練化など、モデリングの詳細については、モデリング編を参照してください。