ユースケースを使用した要求管理の適用

Roger Oberg、Leslee Probasco、Maria Ericsson

Rational Software ホワイト・ペーパー

TP 505 (バージョン 1.4)

Rational®
the software development company

目次

このホワイト・ペーパーは要求管理に不慣れであり (または、多少の知識があり) 要求プロセスの改善に興味がある方に、独自のアプローチを開発するためのフレームワークを提供します。

ユースケースとソフトウェア要求仕様書 (SRS)

読者にとって要求管理のアクティビティーを有意義なものにするため、著者は Rational の Unified Process (RUP) と業界標準の統一モデリング言語 (UML) から特定のドキュメント・タイプとその他の要求管理成果物を選択しました。RUP と UML はいずれもユースケース駆動型のソフトウェア・エンジニアリング・プロセスを推奨しています。

したがって、このホワイト・ペーパーでは、ソフトウェア要求を指定するためのユースケース駆動型手法を説明します。これらのアクティビティーは、ユースケース・モデルとユースケースの代わりに、または追加して、従来のソフトウェア要求仕様書 (IEEE 標準など) と共に使用することも可能です。

プロセスの時代におけるソフトウェアとシステムの開発

1990 年代は、ほとんどのソフトウェアとシステム開発チームにとって、それ以前の自由奔放な時代と違い、プロセスを強化してきた時代となりました。効果的なソフトウェア開発プロセスを評価し、認定するための標準が導入され、普及しました。ソフトウェア開発プロセスに関する多くの書籍や記事、また、ビジネス・プロセス・モデリングとリエンジニアリングに関する資料が発表されました。効果的なソフトウェア開発プロセスの定義と適用を支援するソフトウェア・ツールの数が増えてきています。この十年間、世界経済のソフトウェア依存度が増したため、開発プロセスが強化され、システム品質が向上しました。

では、どうして現在でもソフトウェア・プロジェクトの失敗率は高いのでしょうか。ほとんどとは言わないまでも、多くのソフトウェア・プロジェクトがいまだに遅延や予算超過、品質の問題に悩まされているのはなぜでしょうか。ビジネス、国家経済、日常活動がソフトウェア・システムに依存を深める中で、構築するシステムの品質を向上させるにはどうしたらよいのでしょうか。

その答えは、専門業務に携わる人材、ツール、そしてプロセスにあります。要求管理はソフトウェア開発で進行中の問題に対する解決策として提案されることがよくありますが、この作業分野の実践原則を改善することについて注意が向けられたことはあまりありませんでした。

このホワイト・ペーパーでは、効果的な要求管理プロセスの要素と、それを実装する上で障害となる点を説明します。

要求管理はソフトウェアのみのプロジェクトのほか、ソフトウェアが最終結果の一部になるだけのプロジェクトやまったく含まれないプロジェクトにも同等に適用されます。便宜上、このホワイト・ペーパーでは以後「システム」がこうしたものすべてを指すこととします。ただし、ハードウェアとの組み合わせの有無にかかわらず、要求管理を複雑にするのはソフトウェア開発の抽象的な本質であり、その点がこのホワイト・ペーパーの主題になっています。

なぜ要求を管理するのか

簡単に言うと、システム開発チームはプロジェクトを成功させるために、要求を管理します。プロジェクトの要求を満たすことが成功の定義です。要求管理に失敗すると、こうした目的を達成できる可能性が低くなります。

以下のような最近の文献に事例を見ることができます。

- 1994 年から 1997 年にかけて Standish Group から発表された CHAOS レポートは、プロジェクトが失敗する場合の最も重大な要因は要求に関連するということを証明しました。[1]
- 1997 年 12 月、Computer Industry Daily は Sequent Computer Systems, Inc. の研究結果として、米国とイギリスの 500 人の IT 管理者の 76 パーセントが過去に完全なプロジェクトの失敗を経験していると報告しました。プロジェクト失敗の原因として最も多かったのが「ユーザー要求の変更」でした。[2]

失敗を回避することは要求管理の十分な動機付けになります。また、プロジェクトが成功する可能性の向上と要求管理のその他の利点も、動機付けの理由として考えられます。Standish Group の CHAOS レポートは、要求を適切に管理することがプロジェクトの成功に最も密接に関連していることも証明しました。

要求とは

要求管理を理解する上での最初のステップは共通語彙に関する合意です。Rational は、要求を「(構築する) システムが従わなければならない条件または機能」と定義しています。米国電気電子学会 (IEEE) も同様な定義を使用しています。

著名な要求工学の著者である Merlin Dorfman 氏と Richard H. Thayer 氏は、ソフトウェアに特有の、ただし必ずしも限定されるものではない定義を、矛盾のない、より洗練された形で次のように説明しています。

ソフトウェア要求は以下のように定義できます。

- ユーザーが問題を解決したり、目標を達成したりするために必要なソフトウェア機能。
- 契約、仕様、標準、またはその他の正式に課せられた文書を満たすために、システムまたはシステム・コンポーネントが実現または所有しなければならないソフトウェア機能。[3]

要求管理とは

要求は構築するシステムが従わなければならないものであり、一定の要求に対する適合はプロジェクトの成否を決めることから、要求を理解し、それを文書化し、組織化して、変更時には追跡する必要があります。

要求管理を以下のように言い換えることができます。

- システムの要求を顕在化、組織化、文書化するための体系的手法。
- システムの変更要求に関し、顧客とプロジェクト・チーム間の同意を取りつけ、維持するプロセス。

この定義は、Dorfman 氏と Thayer 氏および IEEE による「ソフトウェア要求工学」の定義に似ています。要求工学には、ソフトウェア要求の顕在化、分析、詳細化、検証、および管理が含まれ、「ソフトウェア要求管理」をこうした一連のアクティビティーを計画し、管理するものと位置付けています。[4] これらのすべてのアクティビティーは、このホワイト・ペーパーで説明し、Rational が提唱する要求管理の定義に組み込まれています。違いは主に用語の選択にあり、「工学」ではなく、「管理」を用います。管理は、関連するすべてのアクティビティーをより適切に説明し、利害関係者とプロジェクト・チーム間の合意を維持するために、変更を追跡する重要性を的確に強調します。

「顕在化」という用語はあまり見慣れないかもしれませんが、これは、利害関係者の要望を見つけ、要求の裏にある本当のニーズを見極めて、ニーズを満たすためにシステム上に配置される適切な要求に到達するために、チームが用いる一連のアクティビティーとして定義されます。

要求管理の問題

では、システムを確実に期待どおりのものにするためにはどのような難しさがあるのでしょうか。現実のプロジェクトではさまざまな困難が明るみに出てきます。図 1 は 1996 年に開発者、管理者、および品質保証担当者を対象に行った調査の結果を表しています。この図は、頻繁に取り上げられる要求関連の各問題に遭遇した回答者のパーセンテージを示します。

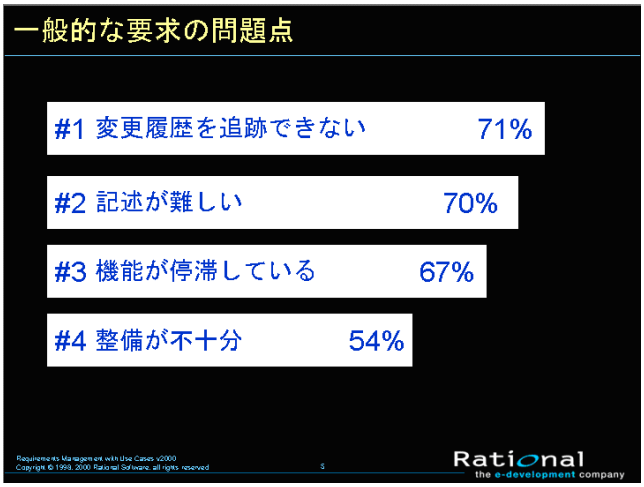

図 1: 一般的な要求問題

さらに、次のような問題点も報告されています。

- 要求は常に明らかなわけではなく、要求元も多数存在する。
- 要求はいつも明快な言葉で表現できるとはかぎらない。
- 多くの異なる種類の要求が、さまざまな詳細なレベルで存在する。
- 要求の数をコントロールしないと、増えすぎて管理不可能になることがある。
- 要求は、別の要求やプロセスの他の納入物とさまざまな点で関連がある。
- 要求には固有のプロパティーまたはプロパティー値がある。例えば、これらの重要性や、達成の難易度は、それぞれ異なったものとなります。
- 関係者や責任者の数が多いので、所属先のさまざまな人によって、要求が管理されなければならない。
- 要求は変化していくものである。
- 要求は時間の影響を受けることがある。

こうした問題に加えて、要求管理とプロセスのスキルが不十分で、使いやすいツールがないと、多くのチームが要求を適切に管理できず、悲観的になります。Rational は要求管理スキルとプロセスについてチームを訓練するための専門技術を開発しました。また、Rational RequisitePro® は効果的な要求管理を簡単に自動化するツールです。

要求管理スキル

これまで述べてきた問題を解決するために、Rational は主要なスキルの開発を促します。以下に示すスキルには一連の順序を示す番号が付いていますが、効果的な要求管理プロセスでは、さまざまな順序で用いられます。ここでは、新規プロジェクトの最初の反復において、一般的に適用される順序を示しています。

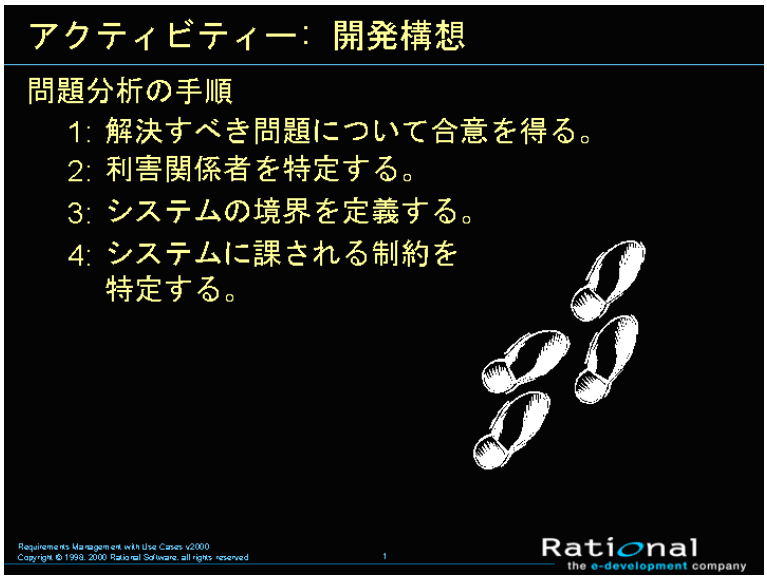

図 2: 問題分析におけるステップ

主要なスキル 1:問題を分析する

問題の分析は、ビジネスの問題を理解し、利害関係者の当初のニーズに的を絞り、高度なソリューションを提案するために行います。こうした推論と分析を行うことにより、「問題の裏に隠れた問題」を見つけることができます。

問題の分析中に、真の問題の記述について合意を形成し、利害関係者を特定します。当初のソリューション境界と制約を技術およびビジネスの観点から定義します。必要な場合は、プロジェクトの開発企画書によりシステムから得られる投資収益率を分析します。

主要なスキル 2:利害関係者のニーズを理解する

要求元は多数存在します。要求はプロジェクトの成果に利害関係を持つ任意の人から提起される場合があります。顧客、パートナー、エンド・ユーザー、ドメインの専門家などは要求元の例です。経営層、プロジェクト・チーム・メンバー、企業方針、監督官庁が要求元になることもあります。

適切な要求元の特定、要求元との連絡、および要求元からの情報の顕在化を行う方法を知ることは重要です。この情報の主なソースとなる人々をプロジェクトの「利害関係者」といいます。

社内で使用する情報システムを開発する場合は、エンド・ユーザーとしての経験者やビジネス・ドメインの専門家に、開発チームに参加してもらうこともできます。非常に多くの場合、システム・レベルではなく、ビジネス・モデル・レベルから検討が開始されます。市販される製品を開発する場合は、対象となる市場における顧客のニーズをよりよく理解するために、マーケティング関係者から助けを得ることもできます。

要求を導き出す技法には、インタビュー、ブレーンストーミング、概念のプロトタイプ化、アンケート、競争的分析などがあります。要求を顕在化させる結果として、テキストやグラフィックで記述された要求やニーズのリストが作成され、互いに相対的な優先度がつけられます。

主要なスキル 3:システムを定義する

システムの定義とは、利害関係者の要望を理解し、構築するシステムに対して意味のあるように記述して整理することです。システム定義の初期に、要求の構成要素、文書フォーマット、言語形式、要求の度合い、要求の優先度と想定される

作業量、技術および管理のリスク、範囲が決定されます。アクティビティーの一部として、最も重要な利害関係者の要望と直接関連する、初期プロトタイプと設計モデルが作成されることもあります。

「文書」という用語は一般的に使用される意味に限定される可能性があるため、代わりに「記述」という用語を使用することにします。記述は、システム自体に不足しているシステム要求を伝達するための手書き文書、電子ファイル、画像、その他の表現です。

システム定義の結果、自然な言葉遣いで、グラフィックを使用したシステムの記述が作成されます。後のセクションでお勧めの記述形式を紹介します。

原則 55:形式的なモデルの前に自然な言葉で書く

正式なモデルを最初に書く場合、ソリューション・システムの代わりにモデルを記述する自然な言葉を書く傾向があります。以下の例を検討してみてください。

> 長距離電話をかけるには、まず受話器を上げます。システムから発信音が聞こえます。「9」をダイヤルします。システムから別の発信音が聞こえます。
>
> システムは、アイドル、発信音、別の発信音、接続という 4 つの状態から構成されます。アイドル状態から発信音状態に移るには、受話器を上げます。発信音状態から別の発信音状態に移るには、「9」をダイヤルします。

後者の例では、テキストがまったく読者の役に立ちません。

—"Alan M. Davis, 201 Principles of Software Development", 1995 (邦訳:「ソフトウェア開発 201 の鉄則」松原友夫訳、日経 BP 社)

主要なスキル 4:システムの範囲を管理する

プロジェクトの範囲は、割り当てられた要求セットにより定義されます。プロジェクトをうまく管理するには、プロジェクトの範囲を利用可能なリソース (時間、人材、予算) に合わせて管理することが大切になります。範囲の管理は継続的なアクティビティーで、プロジェクト範囲を小さな管理しやすい単位に分割し、反復的またはインクリメンタルに開発していきます。

要求の取り込みを交渉する際の基礎として、優先度、作業量、リスクなどの要求属性を使用することは、管理範囲を管理するための非常に有効なテクニックです。要求そのものではなく、属性に焦点を当てることにより、議論になりがちな交渉をうまく進めることができます。

また、チーム・リーダーが交渉スキルに長けていて、組織内および顧客側にプロジェクトの提唱者がいるのも効果的です。製品またはプロジェクトの提唱者には、利用可能なリソースを超える範囲変更について拒否したり、追加範囲を取り込む場合にリソースを拡張したりする決定権限が必要です。

主要なスキル 5:システム定義を洗練する

合意された高レベルのシステム定義と十分に把握された初期範囲を元にして、さらに洗練されたシステム定義を得るためにリソースを投入することが可能であり、同時に経済的です。システム定義の洗練には重要な考慮点が 2 つあります。1 つは高レベルのシステム定義のより詳細な記述を作成することで、もう 1 つはシステムが利害関係者のニーズに合致し、記述されたとおりに動作することを検証することです。

記述はプロジェクト チームにとって重要な参照資料になることがよくあります。優れた記述は読者を想定して作成されます。よくある間違いは構築が複雑なシステムを複雑な定義で表現してしまうことです。 この間違いは、特に読者が合意形成に必要となる重要な思考をできないか、またはそうする意思を持たない場合に発生します。こうなると、プロジェクト・チーム内外の人々にシステムの目的を説明するのが困難になります。場合によっては、読者に応じて何種類かの異なる記述を

用意する必要があるかもしれません。このホワイト・ペーパーには、詳細な自然言語、公式テキスト、グラフィック記述に関する推奨形式が含まれています。記述形式を定めた後、洗練作業はプロジェクトのライフ・サイクルを通じて継続します。

主要なスキル 6:要求変更を管理する

どんなに注意深く定義しても、基本要件は変更されるものです。実際、望ましい要求変更もあります。これはチームが利害関係者を引きつけていることを意味します。要求変更を受け入れることは、利害関係者に対するチームの反応性と運用の柔軟性という成功プロジェクトに貢献するチーム属性を測る尺度になります。変更は敵ではありません。管理されない変更が敵なのです。

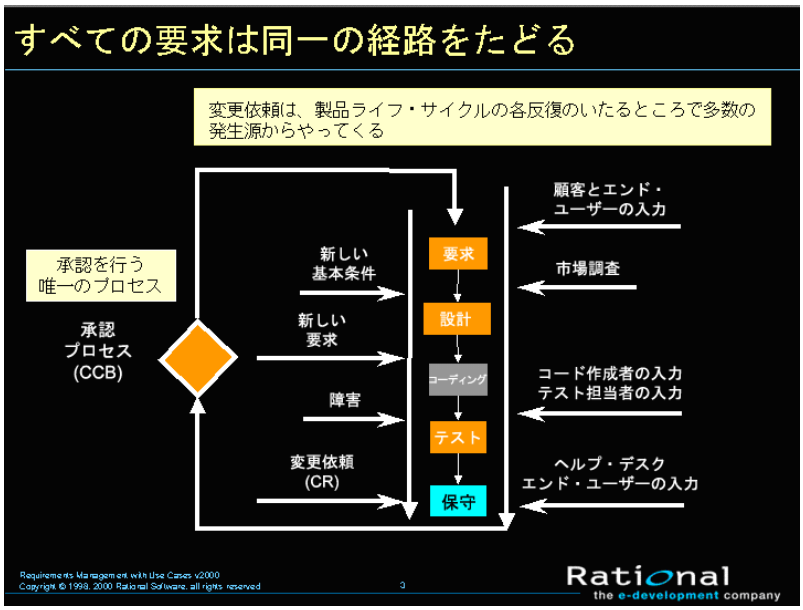

図 3: 変更を管理するプロセス

変更される要求は、特定基本要件の実装に多かれ少なかれ時間を割かなければならないことを意味し、ある要求に対する変更はほかの要求に影響を与えることがあります。要求変更の管理には、ベースラインの作成、各要求の履歴の把握、追跡に重要な依存性の特定、関連項目間の追跡可能な関係の確立、バージョン管理の保守などのアクティビティーが含まれます。図 3 に示すように、変更管理または承認プロセスを確立し、指定されたチーム・メンバーがすべての変更提案を必ずレビューすることも重要です。この変更管理の単一のチャネルを変更管理委員会 (CCB) ということがあります。

重要な要求概念

要求管理スキルをプロジェクトに適用するには、プロジェクトに関わるすべての人が一定の要求管理概念を理解することが有効です。その概念とは以下のようなものです。

- 要求タイプ
- 混成チーム
- 追跡可能性
- 多次元属性
- 変更履歴

要求タイプ

システムの規模が大きくなり、込み入ってくると、要求のタイプも多くなってきます。要求タイプとは単純に要求のクラスです。要求のタイプを識別することにより、チームは多数の要求を、意味があり管理ができるグループにまとめることができます。プロジェクトの異なる要求のタイプの作成は、チーム・メンバーが変更要求を分類し、より明確に伝達するのに役立ちます。

たいていの場合、1 つの要求タイプはほかの複数のタイプに分解できます。ビジネス・ルールと開発構想書の記述は、チームがユーザーのニーズ、基本要件、および製品要求タイプを導き出す高レベルの要求タイプとなることがあります。ユースケースやその他のモデリングの形式は設計要求を促し、その要求をソフトウェア要求に分割し、分析および設計モデルで表現できます。テスト要求はソフトウェア要求から派生し、具体的なテスト手順に分割されます。プロジェクト内に何百、何千、場合によっては何万という数の要求インスタンスがあっても、要求をタイプに分類することでプロジェクトはより管理しやすくなります。

混成チーム

テストやアプリケーション・モデリングなど、ただ 1 つのビジネス・グループの中で管理可能な他のプロセスとは異なり、要求管理には開発プロセスに自分の専門技術で貢献できるすべての人が参加するべきです。顧客および企業の期待を代表する人々は含まれていなければなりません。開発管理者、製品管理者、アナリスト、システム・エンジニア、さらに顧客自身も参加するべきです。また、要求チームには、システム・ソリューションを作成する人、つまり、エンジニア、アーキテクト、デザイナー、プログラマー、テクニカル・ライター、その他の技術者も含める必要があります。テスト担当者などの QA (品質保証) 要員も重要なチーム・メンバーとして数えなければなりません。

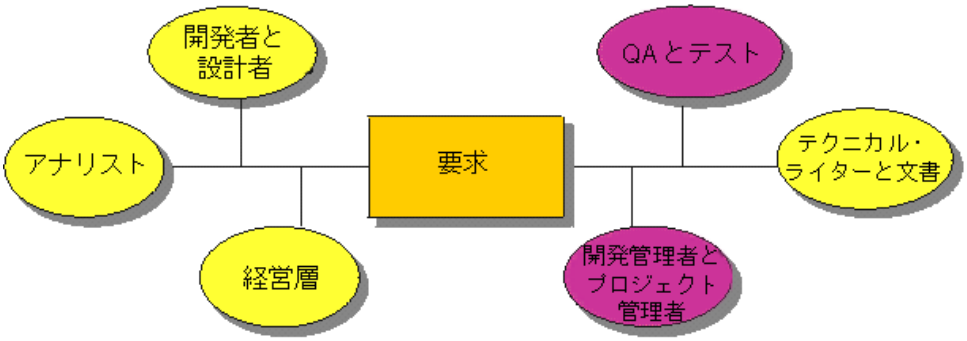

図 4: 混成要求チーム

要求タイプの作成と管理の責任は、機能領域別に割り振ることができる場合が多く、規模の大きなプロジェクトの管理に有効です。このように、さまざまな部署や分野にかかわるという点が、要求管理を難しくしている要因の 1 つです。

追跡可能性

要求タイプの説明からわかるとおり、単体で成立する要求の表現は 1 つもありません。利害関係者の要望はそれを満たすために提案される製品の特徴と関連しています。製品の特徴は、機能および機能外の振る舞いという点で特徴を明確化する個々の要求に関連しています。テスト・ケースは検証対象となる要求に関連しています。要求は、ほかの要求に依存する場合や、相互に排他的な場合があります。チームが変更の影響を判断し、システムを確実に期待に合致させるには、以上のような追跡可能性関係の理解、文書化、および管理が必要です。追跡可能性は要求管理において実装するのが最も困難な概念の 1 つですが、変更の対処には欠かせません。明確な要求タイプを作成し、さまざまな部門の協力を得ることで、追跡可能性の実装と管理は容易になります。要求追跡可能性の異なる戦略に関する詳細については、ホワイト・ペーパー『ユースケースを使用した要求管理の追跡可能性戦略』[5] を参照してください。

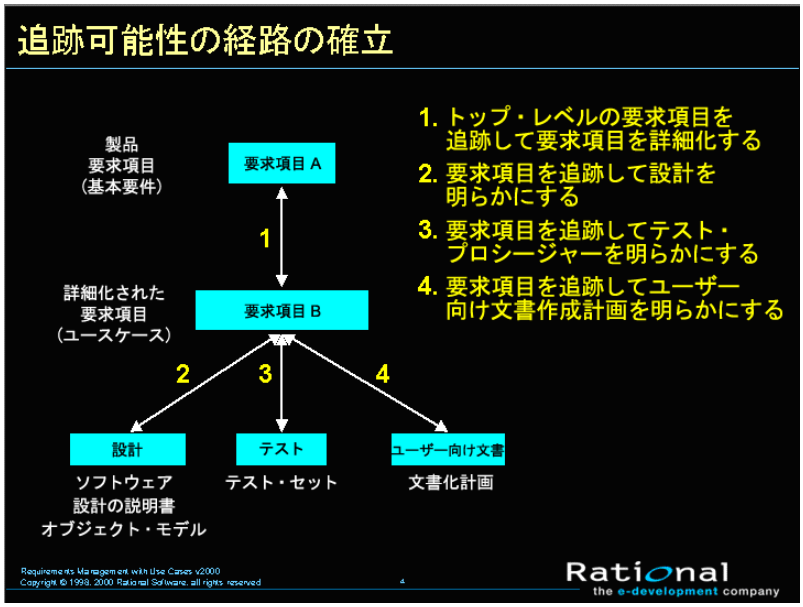

図 5: 要求追跡可能性

多次元属性

各要求タイプは属性を持ち、各個別要求は異なる属性値を持ちます。例えば、要求に対して、優先度の割り当て、ソースと根拠による識別、機能領域内の特定のサブチームへの委任、難易度の指定、システムの特別な反復との関連付けなどが可能です。図 6 は、Rational RequisitePro 要求管理ツールで、学習プロジェクトの基本要件の要求タイプに関する属性を示しています。画面のタイトルから推測できるように、要求タイプと各タイプの属性はプロジェクト全体に対して定義されるので、チーム内で整合性をもって使用することができます。

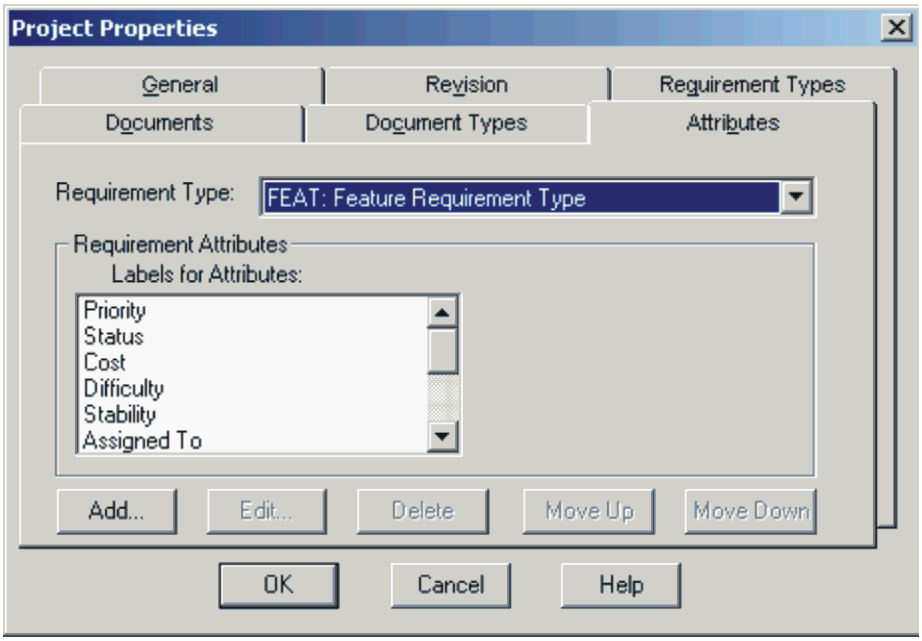

図 6: 基本要件に対する要求属性の定義

図 7 は、RequisitePro で、特定のプロジェクトにおける基本要件の例を示しています。各要求のテキスト全部を表示しなくても、その属性値から各要求の多くを学び取ることができる点に注意してください。この例では、優先度と難易度 (別のチーム・メンバーが割り当てたものだとしても不思議ではありません) により、チームは、利害関係者の優先度と難易度属性値に反映される大まかな作業量の予測を考慮して、プロジェクトの利用可能なリソースと時間への割り当てを開始します。より詳細な要求タイプでは、優先度および作業量属性が、開発範囲を洗練する根拠となる具体的な値 (時間やコード行の見積もりなど) を持つ場合があります。要求におけるこの多次元の側面は、それぞれが固有の属性を持つ異なる要求のタイプから合成され、多数の要求を体系化し、プロジェクトの範囲全体を管理するのに不可欠です。

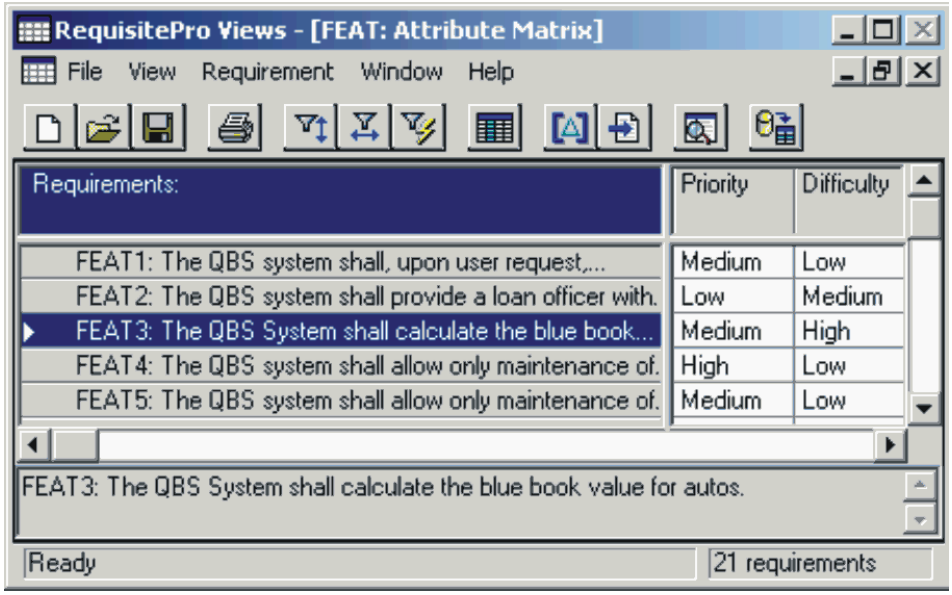

図 7: 各要求の属性値の設定

変更履歴

個々の要求も要求の集合も時間の経過と共に価値が高まる履歴を持っています。環境の変化や技術の進化に追随するために、変更は避けることができず、望ましいものです。プロジェクト要求のバージョンを記録することで、チーム・リーダーは、新規システム・リリースなどのプロジェクト変更に対する理由を把握することができます。要求の集合とソフトウェアの特定バージョンとの関連付けが可能であることを理解すれば、変更をインクリメンタルに管理して、リスクを減らし、マイルストーン達成の可能性を高めることができます。個々の要求が発展する過程で、その履歴を理解することは重要です。つまり、変更の内容、理由、時期、責任者を明確にすることです。

要求管理の実践

要求管理は、以上の主要なスキルと概念を用いて、問題を適切に識別し、解決します。

顧客のニーズを真に満たすシステムを構築するために、プロジェクト・チームはまずシステムにより解決される問題を定義しなければなりません。次に、チームはビジネスおよびユーザーのニーズが顕在化、記述、優先順位付けされる利害関係者を識別しなければなりません。この高レベルの期待つまりニーズのセットから、製品またはシステムの基本要件セットについての合意を形成する必要があります。

詳細なソフトウェア要求は、顧客と開発チームの双方が理解できる形態で作成する必要があります。これまでの経験から、こうしたソフトウェア要求を記述する際に顧客の言葉を使用することが、理解と合意を得る最も効果的な方法であることがわかっています。こうした詳細なソフトウェア要求は、その後、システム設計仕様や、実装と検証に必要なテストの計画および手順の入力として使用されます。ソフトウェア要求はユーザー・マニュアルに関する初期の計画と設計にも役立ちます。

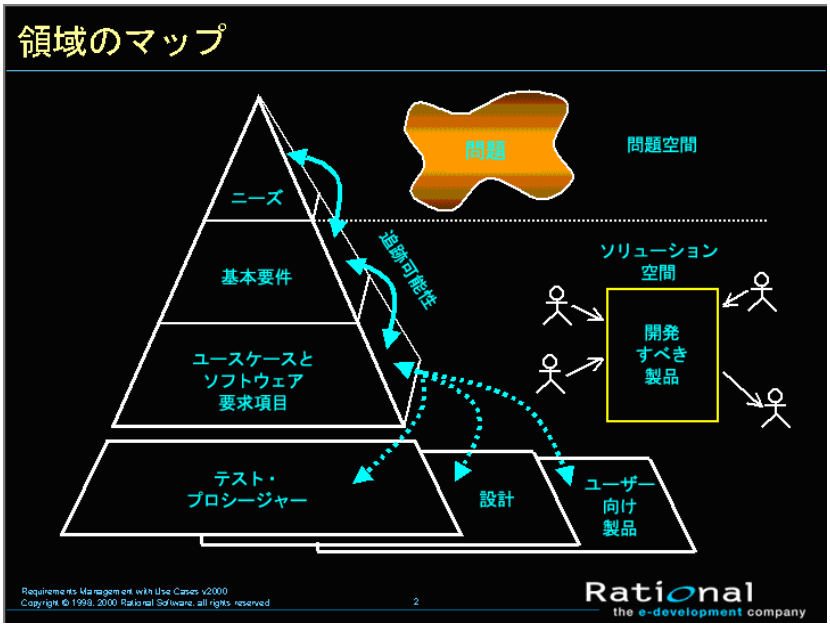

図 8: 要求管理構造の概要

これを実施するには、プロジェクト・チームは以下のことを行う必要があります。

- プロジェクトの共通語彙について合意する
- システムによって解決される問題と主な基本要件を記述するシステム開発構想書を作成する

- 利害関係者のニーズを少なくとも 5 つの重要な領域 (機能性、有用性、信頼性、パフォーマンス、サポート可能性) で導き出す
- 使用する要求タイプを定める
- 各要求タイプの属性と値を選択する
- 要求を記述する形式を選択する
- 1 つまたは複数の要求タイプを作成、貢献、または単に確認するチーム・メンバーを特定する
- 必要な追跡可能性を定める
- 要求に対する変更を提案、レビュー、解決する手順を確立する
- 要求履歴を追跡する仕組みを開発する
- チーム・メンバーと経営層用に進捗および状況レポートを作成する

こうした基本的な要求管理アクティビティーは業界、開発手法、要求ツールには依存しません。また、アクティビティーは柔軟性があるので、最も厳格で、最も短時間のアプリケーション開発環境で効果的な要求管理が可能です。

文書について

要求を文書に記述する決定は考慮する必要があります。一方で、文章の記述は広く受け入れられるコミュニケーション形態で、ほとんどの人にとって自然なことです。他方、プロジェクトの最終目標はシステムの作成であり、文書の作成ではありません。

一般的に考えて、判断の基準は、要求を記録するかどうかではなく、どのように記録するかということです。文書テンプレートは要求管理の統一形式を提供します。Rational RequisitePro はこうしたテンプレートの他、文書内の要求をすべてのプロジェクト要求を含むデータベースに連携させる機能も提供します。この特別な機能により、要求を自然な言葉で記述しつつ、リレーショナル・データベースで利用や管理をしやすくします。

要求管理: アクティビティー

要求管理がたどる可能性があるドメインごとの経路は数知れません。以下に述べる手法は 6 つの詳細なアクティビティーを規定します。これらのアクティビティーは主要な要求管理スキルのそれぞれに適用されますが、任意のドメインに適用することもできます。

以下の各アクティビティー図は Rational Unified Process [6]要求の作業分野から抜粋しています。これらのアクティビティーは、図 9 にまとめたロール、タスク、成果物 (入力または出力) という点から表現しています。また、このホワイト・ペーパー内の付随テキストは各アクティビティーを簡単に説明しているため、読者はこれらを参考にして、独自の要求管理プロセスを改善できます。Rational Unified Process について詳しくは、www.ibm.com/software/rational に記載しています。

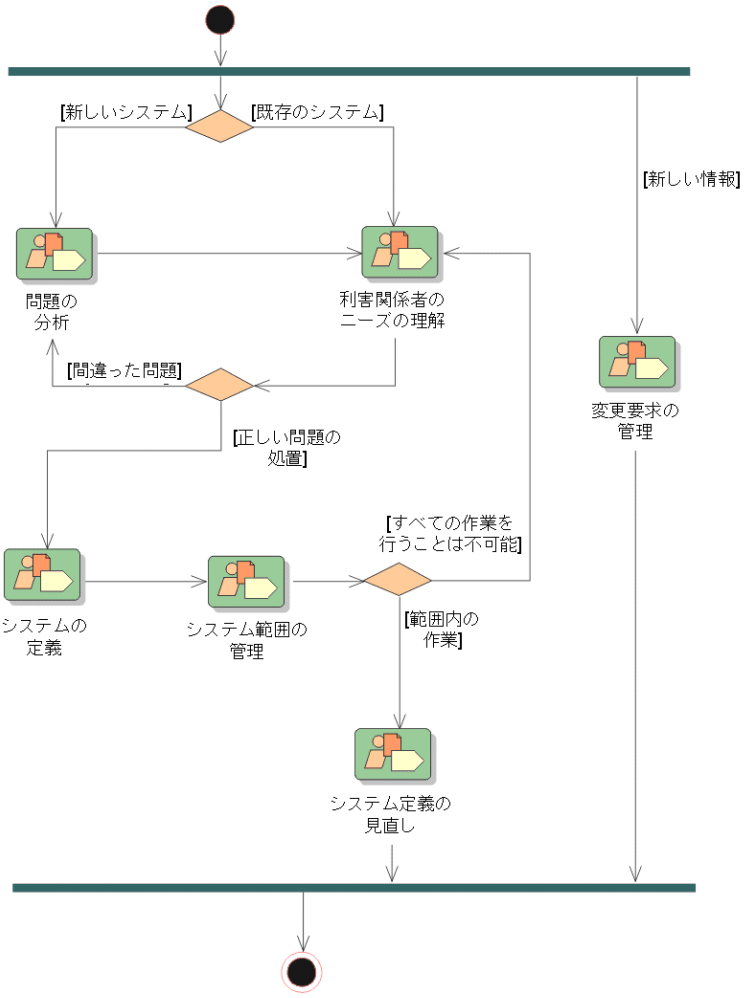

図 9: Rational Unified Process における要求ワークフロー

アクティビティー:問題を分析する

問題分析における主なアクティビティーはプロジェクトの開発構想書を作成することです。このアクティビティーの目的は、構築するシステムをハイレベルなユーザーまたは顧客から見た場合の開発構想書を作成することです。開発構想書は、最も重要な問題を解決し、主要な利害関係者のニーズを満たすためにシステムが持たなければならない主な基本要件として初期の要求を表現します。システム分析者がこのアクティビティーで主要なロールを持ちます。システム分析者は、問題の該当領域の専門技術と知識を持っていなければならず、また、問題を解決すると自らが考えるプロセスを記述できなければなりません。さまざまなプロジェクト利害関係者が積極的に参画することが求められ、関連するすべての利害関係者の要望を考慮する必要があります。

依存性タスクの管理を開始するには、根拠、相対値または優先度、要求元といった属性を基本要件に割り当てなければなりません。開発構想が具体化する中で、アナリストは可能性があるユースケースのユーザーとシステム (アクター) を識別します。アクターはユースケース モデルの最初の要素であり、システムの基本要件および機能以外の技術要求を定義します。

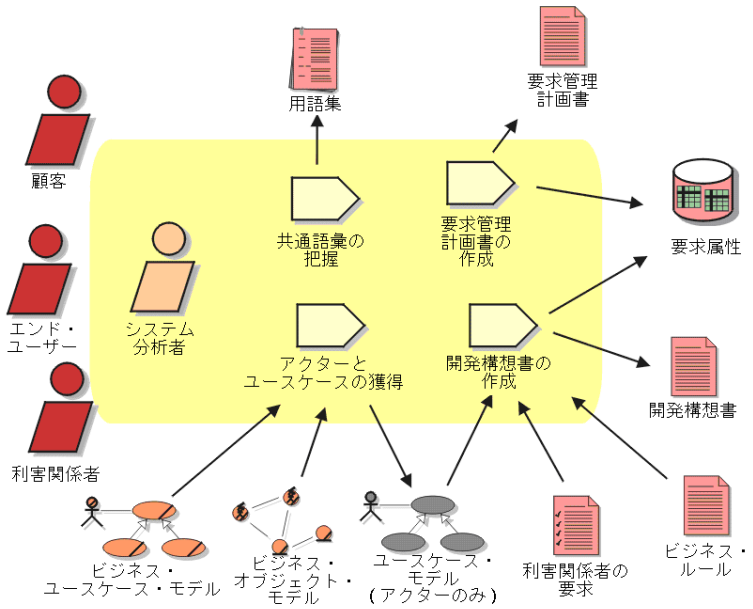

図 10: 問題の分析

アクティビティー:問題を分析する

開始:問題を認識する 1 人以上の利害関係者がアクティビティーを開始します。

開発チーム内のシステム分析者は、会議を進行して、当初の利害関係者が解決したい問題を記述するのを支援することができます。認識された問題の簡潔な記述について合意を得ることが重要です。**問題の記述**の主要な要素を以下の表に示します。

問題の内容	問題を定義する
影響を受ける利害関係者	問題によって影響を受ける利害関係者を列挙する
問題の影響	問題の影響を記述する
適切な解決策	適切な解決策の主な利点を列挙する

問題の記述はプロジェクトの目的を簡潔に説明します。問題分析はすべての利害関係者の要望調査の他、説得力のある利点や概算費用を含む初期開発企画書を促します。問題の記述の定義と並行して、チームは一般的に使用される用語を把握し、それらの定義について合意することにより、用語集の編集も行う必要があります。

ユースケース・モデルの概説

ユースケース・モデルはアクター、ユースケース、およびそれらの間の関係から構成されます。アクターは、システムと情報を交換しなければならないすべてのものを表します。一般的な意味のユーザーもアクターです。アクターがシステムを使用するとき、システムはユースケースを実行します。適切なユースケースはアクターに対して値の測定可能な結果を出力する一連のトランザクションです。ユースケースの集合はシステムの完全な機能となります。

—Jacobson I.、Christerson M.、Jonsson P.、Overgaard G.、"Object-Oriented Software Engineering – A Use Case Driven Approach, Addison Wesley" – ACM Press、1992 年 (邦訳:「オ

ブジェクト指向ソフトウェア工学」西岡利博他監訳、トッパン)

問題分析は主なシステム・アクターも識別します。アクターは、開発対象システムまたは開発対象システムと情報を交換する任意の他のシステムのユーザーです。この段階で、問題分析はアクターがシステムと相互作用する明白な方法を簡単に定めなければなりません。記述は、システムの動作ではなくビジネス・プロセスの視点で行わなければなりません。例えば、予算編成プログラムはあるタイプのアクターに「部門予算を作成する」ことを許可し、別のアクターに「部門予算を統合する」ことを許可する場合があります。その後、システム分析者はこうしたユースケースを、補足的なユースケースに分割して、特定のシステム動作とより有意義に連携させることができます。たとえば、「部門予算を作成する」は最終的に「スプレッドシート情報をインポートする」や「予算ビューを作成する」といったシステム・ユースケースになることが考えられます。

上で述べた問題分析会議は複数回行われることが多く、別の利害関係者が出席したり、内部の開発チームの会議と合同で行ったりすることもあります。利害関係者との会議を行ったシステム分析者は開発チームのメンバーとの会議を先導し、問題に対する技術的な解決策の構想策定、当初の利害関係者による入力情報からの基本要件の導出、構築するシステムの最初の定義となる開発構想書の草案作成を行います。提案した解決策を当初の利害関係者に十分理解してもらうために、システム分析者はモデリング・ツールや手書きの図解手法を使用して開発構想書の記述を補足することができます。

さまざまな時点で当初の利害関係者の意見を聞き、問題記述を洗練し、可能な解決策の数と範囲を絞り込みます。利害関係者とシステム分析者は、主な基本要件の優先度を交渉し、それらを開発するために必要となるリソースと作業量について全般的な理解を得ることで、このアクティビティー内の依存性を管理します。優先度と作業量、またはリソースはどうしても変わるものですが、初期段階で依存性を管理することにより、開発ライフ・サイクルを通じて継続する重要なパターンが作成されます。これは開発範囲管理の根本であり、プロジェクトの成否を早い時期に予測します。

開発構想書は、いくつかの草稿を経た後、チームが追加の要求顕在化に投資すべきかどうかを決めなければならない点に達します。同じ時点までに、開発企画書の承認プロセスが別個に開始されます。このホワイト・ペーパーでは詳細を述べませんが、開発企画書は以下のすべてを記述します。

- 内容 (製品分野、市場、および範囲)
- 技術的な手法
- 管理手法 (スケジュール、リスク、成功の客観的な測定)
- 財務予測

アクティビティー:利害関係者のニーズを理解する

初期の問題分析で追加投資が認められると、利害関係者のニーズの理解のアクティビティーが正式に開始します。ここでの主なタスクは利害関係者の要求を導き出すことです。主要な出力は優先順位付けされた利害関係者のニーズと基本要件の集合であり、これらにより開発構想書が洗練され、要求属性をより深く理解することが可能になります。また、このアクティビティーの中で、ユースケースとアクターに焦点を当てたシステムの議論を開始できます。このアクティビティーによるもう 1 つの重要な出力として、最新の用語集を作成し、チーム内で用語を統一できるようにします。

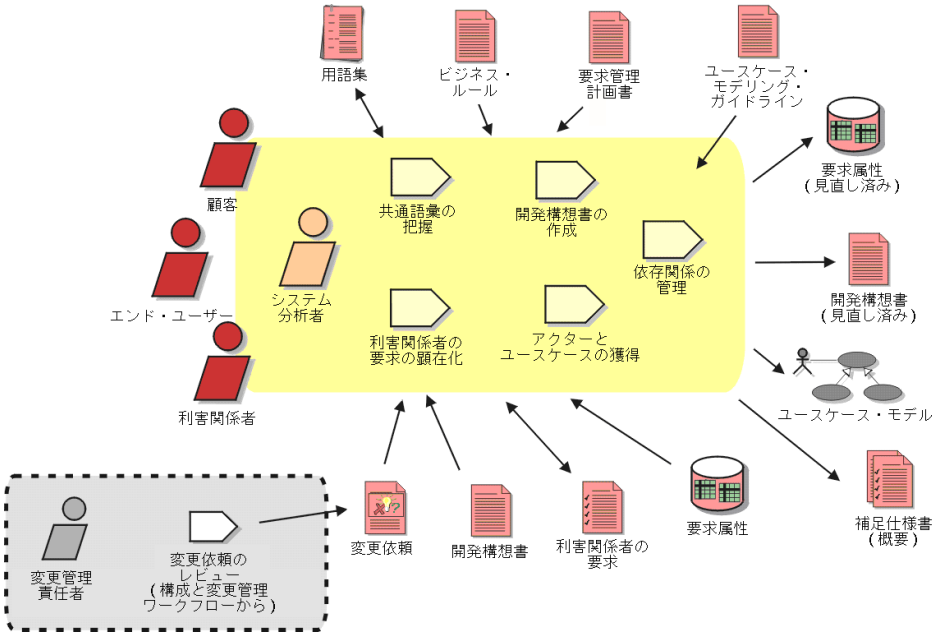

図 11: 利害関係者のニーズの理解

アクティビティー:利害関係者のニーズを理解する

システム分析者と主な利害関係者は、追加の利害関係者を特定し、インタビュー、検討会議、絵コンテ、ビジネス・プロセス・ユースケース、その他のテクニックを用いて、要求を導き出し、主要なニーズと基本要件を識別します。1 人以上のシステム分析者がこうした会議を推進します。こうした要求検討会議は最も有効な顕在化テクニックの 1 つです。このプロセスには、ユーザー、ヘルプ・デスク要員、経営者、テスト担当者、提案されるプロジェクトの成果に利害関係を持つその他の人々が含まれます。利害関係者の要望はあいまいだったり、重複していたり、時には無関係だったりします。利害関係者の要望は、正式な顕在化結果の他、適正な書式の文書、データベースからの欠陥と拡張要求、または電子メールやグループウェアのスレッドという形態で表される場合があります。システム分析者は識別された主要な利害関係者のニーズを記録、分類、優先順位付けします。

利害関係者のニーズを理解する:「顧客満足」の出発点

利害関係者の要望は、要求元の利害関係者が用いる言葉と形式において可能な限り把握されます。一般に、プロセス教育を受け、技術的に熟達したプロジェクト・チーム・メンバーにより作成される要求タイプと異なり、利害関係者の要望は表現が稚拙な場合がよくあります。複数の要求が重複していることもあります。また、紙片から拡張要求データベースまで、さまざまな形で表現されます。

アナリスト (または分析のロールを持つチーム) は、そうしたすべての要求をレビューしなければなりません。要求の解釈、グループ化、場合によっては再入力 (再作成ではない) を行い、開発構想書内の主要な利害関係者ニーズとシステム基本要件に変換する必要があります。開発に求められる厳密さとツールの機能に応じて、利害関係者の要望、ニーズ、および基本要件の一部またはすべてに関する追跡可能性を適用することにより、利害関係者はシステムの要求を定める上で自らの要求とニーズがどのように説明されるかを理解しやすくなります。

利害関係者のニーズの理解のアクティビティーを適用して、要求の顕在化と利害関係者のニーズの充足への重大な関心を提示することは、開発チームの能力に対する利害関係者の信頼を確立する点で重要になる場合があります。

利害関係者ニーズのより正確な理解に基づき、開発チームのシステム分析者は開発構想書を洗練します。この際、「製品の位置付け記述」を作成することに特別な注意を払います。この数行の記述はプロジェクトに説得力のある価値を生みます。記述には、想定されるユーザー、解決される問題、製品がもたらす利点、置き換わる競合製品が含まれていなければなりません。すべてのチーム・メンバーはこのプロジェクト テーマを理解しなければなりません。システム分析者は用語集を更新して、用語の共通理解を促進することも必要です。

主な利害関係者とはさまざまな時点で連絡を取り、利害関係者のニーズの理解から顕在化した新しい基本要件の優先度について交渉し、それらを開発するために必要なリソースと作業量に関する現在の理解を得ます。問題分析の場合と同様に、このアクティビティーにおける依存性の管理は開発範囲を管理するのに役立ちます。また、依存性の管理は利害関係者の要望、ニーズ、およびシステム基本要件の間に追跡可能性を確立するため、利害関係者は自らの入力情報が考慮されていることを確認できます。

アクティビティー:システムを定義する

最初の 2 つの要求アクティビティーである問題の分析および利害関係者のニーズの理解は、主要なシステム定義の初期の反復を作成します。これには、開発構想書内で指定される基本要件、ユースケース・モデルの最初の概要、および初期要求属性が含まれます。次のアクティビティーであるシステムの定義は、高レベルのシステム要求の記述を、基本要件定義の洗練、新しいアクターやユースケース、補足仕様書の追加を行うことにより完成させます。

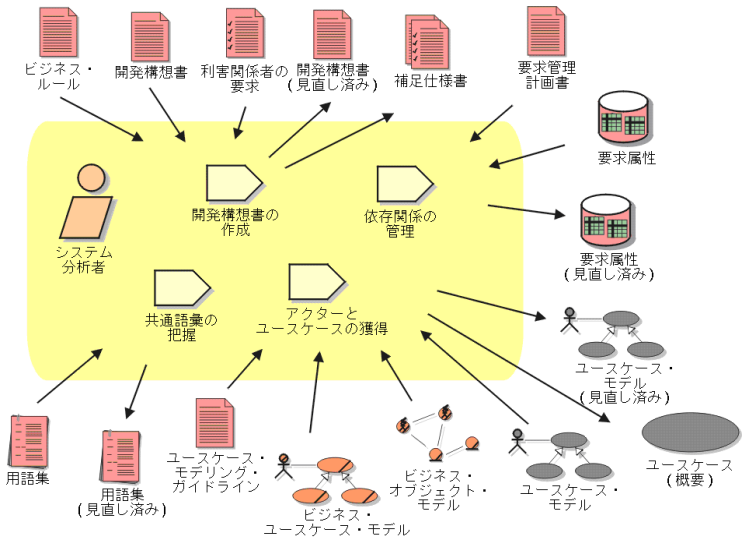

図 12: システムの定義

アクティビティー:システムを定義する

ユースケース・モデルと補足仕様書で把握された基本要件と要求を説明するために使用される用語について、現在の理解を反映するように用語集が更新されます。

システム分析者は、洗練された開発構想書内で定義された基本要件セットを使って、ユーザーの視点で見たシステムに期待される動作を説明するユースケースを導き出し、記述します。このユースケース・モデルは、選択された特性がシステム内でどのように機能するかに関する、顧客、ユーザー、およびシステム開発者間の契約となります。このモデルは、システム開発者にとっては現実的な期待と目標を設定するのに役立ち、顧客とユーザーにとってはシステムが期待に合致するかどうかを検証するのに役立ちます。

システム要求の中にはユースケースに適さないものもあります。システム分析者はこれらを補足仕様書に記述します。使いやすさ、信頼性、性能、サポートのしやすさといった機能外要求の多くはここに含まれる場合がよくあります。こうしたタイプの機能外要求の多くはただ 1 つのユースケースに固有であることに注意してください。ユースケース定義者は、これ

らの要求をユースケース仕様そのもの (システムの洗練アクティビティーを参照) に配置し、補足仕様書をシステム規模の機能外要求のために残しておくことが適切です。

このアクティビティーにおいて、システム分析者は、補足要求の属性 (優先度や関連するユースケースなど) を作成し、設定します。さらに、システム分析者は当初および新規のユースケースについて、属性値を追加および更新します。

最後に、システム分析者は、補足仕様書に記述される関連のユースケースと要求に対する重要なユーザー・ニーズと基本要件を追跡することにより依存性を管理します。

アクティビティー:システムの範囲を管理する

基本要件レベルの要求を特定し、ほとんどのアクター、ユースケース、および補足仕様書に指定されたその他の要求を記述した後、システム分析者は、優先度、作業量、費用、リスクといった要求属性について、可能な限り正確な値を収集し、割り当てなければなりません。これにより、システム・リリースの初期範囲を定める方法を十分理解することが可能になり、アーキテクトがアーキテクチャー上重要なユースケースを識別することもできます。

プロジェクトおよび開発管理により、並行して開発される反復計画は、このシステム範囲の管理アクティビティーで最初に現れます。反復計画は、開発計画ともいい、リリースのために計画される反復の数と頻度を定義します。範囲内の最も高いリスク要素は初期の反復で計画しなければなりません。

範囲の管理アクティビティーからの他の重要な出力には、ソフトウェア・アーキテクチャー説明書の初期反復や、システム機能とプロジェクト・リソースに関するアナリストと主要な利害関係者の理解が深まったことを反映する改訂された開発構想書などがあります。

前述した開発企画書や最初に発行される反復計画書と同様に、ソフトウェア・アーキテクチャー説明書は Rational Unified Process の一部であり、関連するものですが、要求管理の成果物ではありません。これはこのホワイト・ペーパーのテーマではありません。

> 経験上、開発範囲の管理を成功させる鍵は、補足仕様書に指定される利害関係者のニーズ、基本要件、ユースケース、その他の要求に十分考慮された属性値を割り当てることです。 同時に、代表的な利害関係者と定期的にオープンで率直な対話を行うことも必要です。

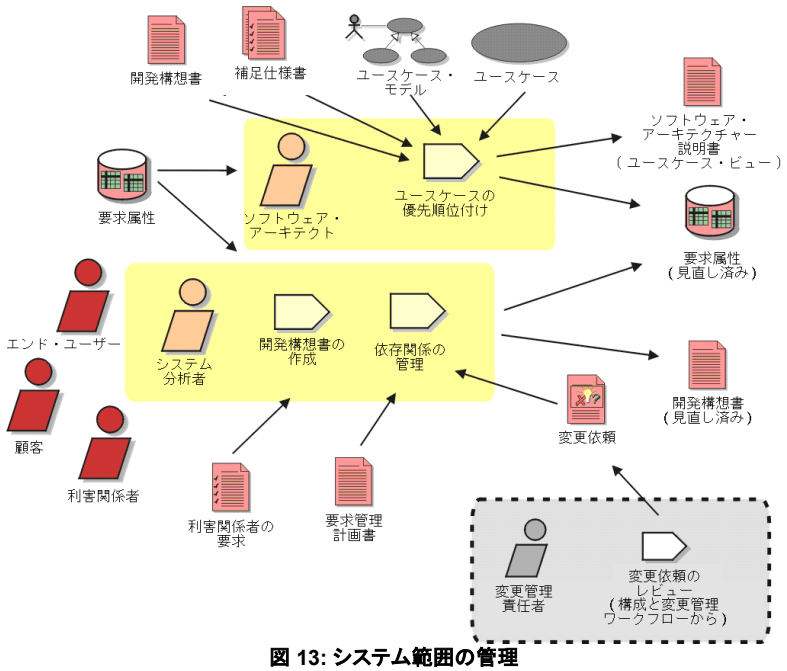

図 13: システム範囲の管理

アクティビティー:システムの範囲を管理する

アーキテクトは、リスク・カバレッジ、アーキテクチャー上の重要性、アーキテクチャー・カバレッジについて、ユースケースを優先順位付けします。システムは多数のユースケースと補足仕様要求で定義されることがありますが、通常、ユースケースのあるサブセットのみがすぐれたシステム・アーキテクチャーに重要です。優先順位付けされたユースケースを元に、アーキテクトは反復または開発計画を洗練し、Rational Rose などのツールでシステム・アーキテクチャーのユースケース・ビューをモデリングします。

システム分析者は、開発構想内の基本要件の要求属性を洗練することにより依存性を管理します。システム分析者はユースケースと補足仕様書内の要求も洗練します。システム分析者は、利害関係者のニーズ、基本要件、ユースケース要求、補足仕様要求に対する適切な追跡可能性* が存在することを確認します。「適切な追跡可能性」という用語は意図的なものです。このホワイト・ペーパーで後述する追跡可能性に関する文書を参照してください。

このステップはプロジェクト全体の中で最も重要なものの 1 つです。初めに、提案されるシステムに関する知識の幅広さを利用して、要求、プロジェクト・リソース、納品日に真剣に取り組むことができます。同時に、こうした要求は知識の深さが増すにつれて変化することを理解しなければなりません。前述した 3 つのアクティビティーで依存性が管理されている場合、このステップははるかに簡単で、今後の変更も容易になります。

プロジェクトが進展する間に、状況と環境が変化すると、システム分析者が改訂されたプロジェクト開発範囲と構想について主要な利害関係者と交渉する必要があるため、このアクティビティーは何度も使用されます。プロジェクトの開発範囲を管理して利用可能なリソースに一致させることは、利害関係者と開発チーム・メンバーがこのステップを自然な進捗として見る場合にだけ成功します。ユーザーの期待を叶えたり、組織に対して時間と予算の追加を強要すると失敗します。このアクティビティーをプロジェクト内の主なマイルストーンで繰り返し、システムとその問題への新しい洞察が開発範囲の変更を必要とするかどうかを評価する必要があります。約束した要求、予算、締め切りを変更するのは困難ですが、優先順位付けされたユースケース、補足仕様書、および初期のシステム反復を徹底的に理解すると必然的に開発範囲を再検討することになります。

繰り返しになりますが、洗練段階に至る前に、また、プロジェクトのライフサイクルを通じて変更が発生する際に、プロジェクト チームは習慣的に開発範囲管理を行うことが重要です。代表的な利害関係者は、ますます難しくなる範囲交渉の中で、自らの優先度と関心が真剣に取り組まれていることを理解し、信頼しなければなりません。システム要求が洗練されるまでに、重要な要求のみが交渉または変更されない状態になります。効果的な開発範囲管理の習慣が確立されていなければ、プロジェクトは「デス・マーチ (システム開発現場の過酷な労働環境を表す)」を余儀なくされ、どうしようもなく開発範囲が広がったプロジェクトは無情にも遅延と予算オーバーへの道を突き進むことになるでしょう。

アクティビティー:システム定義を洗練する

システム定義の洗練に進みますが、このアクティビティーの詳細では、システム・レベルのユースケースが略述され、アクターが少なくとも簡単に記述されていることを前提とします。プロジェクト開発範囲の管理を通じて、開発構想内の基本要件は再度優先順位付けされてきており、この時点ではかなり安定した予算と日数で達成されるものと考えられます。このアクティビティーの出力は、詳細なユースケース、改訂された補足仕様書、システム自体の初期反復で表現されるシステム機能のより深い理解です。

明らかに、すべてのシステムがユーザー・インターフェースを持つわけではなく、すべての初期反復が GUI 要素を含むわけではありません。ここではそれらを初期反復の例としてのみ使用します。その他の例にはプロトタイプ、モデル、絵コンテがあります。

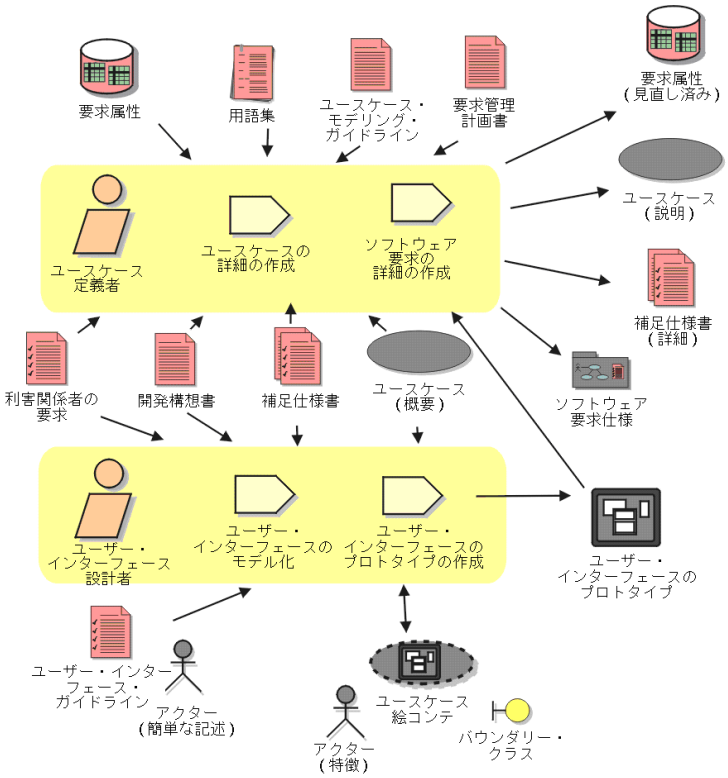

図 14: システム定義の洗練

アクティビティー:システム定義を洗練する

ユースケース定義者は各ユースケースについて、イベントのフロー、事前と事後条件、その他のテキスト形式プロパティーの定義を洗練します。作業量を最小化し、読みやすくするためには、標準の文書形式を使用するか、またはユースケース仕様書テンプレートを使用して、各ユースケースに関するテキスト情報を得ることをお勧めします。十分に考慮されたユースケース仕様書を作成することはシステムの品質に不可欠です。この仕様開発にはユースケースに関連する利害関係者のニーズと基本要件を徹底的に理解することが必要です。できれば、プロジェクト・チームのメンバーのうち数人 (ソフトウェア・エンジニアなど) にユースケースの作成に参加してもらうとよいでしょう。

並行して、ユースケース定義者は補足仕様書をユースケースに固有でない追加要件と共に改訂します。

ユーザー・インターフェースの設計者はシステムのユーザー・インターフェースをモデリングし、プロトタイプを作成します。この作業はユースケースの発展と密接に関連しています。

ユースケース定義者とシステム分析者は、理解が深まった各要求について、作業量、費用、リスク、その他の属性値を改訂します。

このシステム定義洗練の結果は、もう 1 つの別の範囲の管理アクティビティーに送られます。システムの詳細を理解した後、優先度を変更するとよいでしょう。必要な場合は、絶対にシステム・リリースの範囲をレビューし、洗練しなければなりません。

アクティビティー:変更要求を管理する

プロジェクトが進展する間に変更が発生した場合 (変更は避けられません)、システム範囲の管理で述べたように、変更要求の管理アクティビティーを継続的に適用する必要があります。このアクティビティーの出力によりすべての成果物の修正が生じる場合があり、その結果、すべてのプロジェクト・チーム・メンバーと利害関係者間で効果的な意思疎通が必要となります。

このアクティビティーで、要求アクティビティーにより影響を受ける追加の成果物を導入します。要求に対する変更は、分析アクティビティーと設計アクティビティーで表されるシステム・モデルに当然影響します。要求変更は、要求の適切な実装を検証するために作成されたテストにも影響します。前に示した例と同様に、これらの成果物は Rational Unified Process の一部ですが、このホワイト・ペーパーのテーマではありません。依存性を管理するプロセスで識別される追跡可能性関係はこれらの影響を理解する上での鍵になります。

追跡可能性

要求の分野では追跡可能性からさまざまなものが得られます。多くの人々が、個々の顧客要求から各関連仕様、テスト、モデル要素、最終的にはソース・コード・ファイルに至るまで追跡することの利点を推進しています。確かに、いくつかの追跡可能性は要求変更管理の成功の鍵です。

ただし、どの形式の追跡可能性も、プロジェクトのライフを通じてセットアップし、管理するために投資を必要とすることに注意してください。すべての投資と同様に、追跡可能性には特定の状況によって収益の減少点が存在します。このホワイト・ペーパーでは、異なる要求タイプ間での追跡の価値を強調します。手始めに Rational RequisitePro、Rational Rose、Rational SoDA、Rational TeamTest といったツールで自動化することができます。一定のレベルの要求追跡可能性が適切な投資であることを理解できると思います。

要求追跡可能性の異なる戦略に関する詳細については、ホワイト・ペーパー「ユースケースを使用した要求管理の追跡可能性戦略」[6] を参照してください。

変更要求の管理についてのもう 1 つの重要な概念は要求履歴追跡です。要求変更の性質と根拠がわかれば、レビュー担当者 (ソフトウェア・プロジェクト・チームのメンバーで、変更により自分の作業が影響を受ける任意の人) がこの変更に対応するために必要な情報を受けることができます。

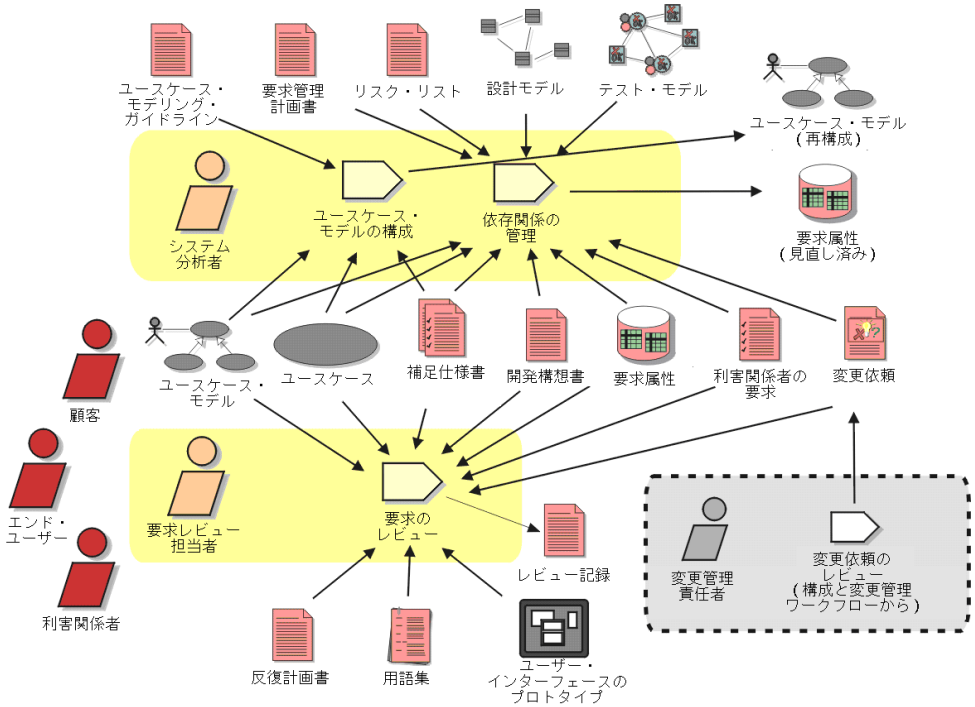

図 15: 変更要求の管理

アクティビティー:変更要求を管理する

要求変更の依頼は、任意の利害関係者またはプロジェクト・チーム・メンバーにより、さまざまな理由から出される場合があります。要求の変更、拡張依頼、欠陥を含むすべての変更依頼 (CR) は、同じ変更依頼管理 (CRM) プロセスを通らなければなりません。最低限、集中管理のデータベースで依頼を記録および追跡し、中枢的なレビュー組織によるレビューを必須としなければなりません。CRM プロセスなどの詳細については、Rational Unified Process の他のセクションを参照してください。

システム分析者は、可能であれば変更管理委員会 (CCB) により、レビュー・アクティビティーを調整し、すべての変更依頼を収集およびレビューして、それらを以下のように分類しなければなりません。

- 要求に影響しない実装中の欠陥
- あるタイプの既存要求に対する変更
- 拡張要求

分類した後、要求に対して提案される変更には、ほかの要求のアクティビティーで述べたように属性と値が割り当てられます。

変更依頼をレビューする際、システム分析者は、代表的な利害関係者とプロジェクト・チーム・メンバーで構成される CCB に対して、優先順位付けした要求変更の提案を示します。リソースを超過する範囲変更は却下されるか、確約された日付と予算について要求変更の承認権限を持つ利害関係者の代表に提出される必要があります。

CCB は要求への変更を承認または却下します。

システム分析者は、要求変更を要求定義者に伝えるか、または開発構想書、ユースケース、補足仕様書、その他の要求成果物の中で直接変更を行います。

要求のレビュー担当者 (開発者、テスト担当者、管理者、その他のプロジェクト・チーム・メンバー) は、要求履歴をレビューすることにより、要求への変更が自らの作業に与える影響を評価します。最後に、レビュー担当者は変更を実装し、権限を持つ関連要求に対して適切な変更を行います。

まとめ

要求を管理するニーズは新しいものではありません。では、これまで述べた情報を今、考慮する価値はどこにあるのでしょうか。

まず、プロジェクトが常に顧客を満足させ、締め切りに間に合い、予算内に収まるわけではない場合、開発手法を再検討する理由があります。再検討した結果、要求関連の問題が開発作業に悪影響を与えていることが判明した場合、より適切な要求管理の実践原則を検討する必要があります。

第 2 に、このホワイト・ペーパーにまとめた要求管理の実践原則は何千という経験を具体化したものであり、要求管理の分野で何年も顧客と作業を行ってきた多くの個人の意見を十分に反映しています。そうした人々の貢献の概要や、Rational Unified Process の中で実行される、より詳細な表現は、要求管理における「最高の実践原則」を表すものとして提案します。要求についてこれ以上実証された助言はほかのどこからも得られないでしょう。

著者は、Ivar Jacobson 博士、Dean Leffingwell 氏、Alan Davis 博士、Ed Yourdon 氏、Elemer Magaziner 氏 の直接および間接の貢献に対して感謝したいと思います。最も重要なこととして、これらの実践原則を何百という開発プロジェクトに適用し、改善してきた Rational の顧客を称賛します。

最後に、この 2 年間、効果的なソフトウェア開発ソリューションのビジネスにおけるリーダーである Rational は要求管理という課題に取り組み、この困難な作業を自動化するツールを開発しました。習慣的で広範囲な要求管理の問題は解決できます。また、究極的に、問題の存在が要求管理におけるすぐれた実践原則を今日開始するための最高の理由になりえます。

以上の概念に関する詳細については、ソフトウェア要求の管理についての Dean Leffingwell 氏の好著 [7].を参照してください。

参考資料

[1] "CHAOS", The Standish Group International, Inc., Dennis, MA, 1994, 1997.

[2] "Computer Industry Daily", December 12, 1997.

[3] Dorfman, M. and R. Thayer, "*Software Engineering*", IEEE Computer Society Press, Los Alamitos, CA, 1997 . 79. 79.

[4] Dorfman, M. and R. Thayer, "*Software Engineering*", IEEE Computer Society Press, Los Alamitos, CA, 1997 . 80.

[5]「ユースケースを使用した要求管理の追跡可能性戦略」、ホワイト・ペーパー、Ian Spence、Leslee Probasco 共著、Rational Software Corporation、1998.

[6] "Rational Unified Process®", Rational Software Corporation, Cupertino, CA, 1999.

[7] Leffingwell, Dean and Don Widrig, "*Managing Software Requirements — A Unified Approach"*, Addison-Wesley, 2000. (邦訳:「ソフトウェア要求管理 新世代の統一アプローチ」日本ラショナルソフトウェア訳、石塚 圭樹、荒川 三枝子 監訳)

Rational®
the software development company

Dual Headquarters:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail:info@rational.com
Web:www.rational.com
International Locations:www.rational.com/worldwide