修 士 論 文

共同ソフトウェア開発における開発者の依存関係に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

周　翼

2003 年 9 月

修 士 論 文

共同ソフトウェア開発における開発者の依存関係に関する研究

指導教官　落水 浩一郎 教授

審査委員主査　落水 浩一郎 教授
審査委員　片山 卓也 教授
審査委員　篠田 陽一 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

110202 周　翼

提出年月: 2003 年 8 月

概 要

ソフトウェア開発においては、開発者が従わなければならない開発プロセスに加えて、開発者間のコミュニケーションの支援が重要である。Rational Unified Process(RUP) は作業内容と作業内容の順序を定義している成果物中心の方法論であるが、コミュニケーションの支援についてはふれていない。一方、Coplien は組織構造のあり方をコミュニケーションパスの構成法の観点から論じている。本論文では、RUP プロセスを、役割の管理モデルとして再定義することにより RUP と Coplien の組織パターンを結合したプロセスモデルを提案する。このプロセスモデルは、成果物に変更があったときに必要となるコミュニケーションとコミュニケーションが発生したときに見直すべき成果物の両者を表現できる。

目 次

第1章　はじめに

1.1　背景

複数の開発者が成果物（図面、文書など）を共有した共同ソフトウェア開発が注目されている。共同ソフトウェア開発を成功させるために、開発者は決められた開発プロセスに従い、成果物を作成しなければならない。

開発プロセスとしては、ラショナル統一プロセス（Rational Unified Process:RUP）が開発現場によく使われている。RUP プロセスとは、米国ラショナルソフトウェア社がBooch、Rumbaugh、Jacobson の３人のそれぞれの手法に加えて、業界におけるさまざまなプラクティスを反映し、成果物中心の開発方法論である。RUP プロセスは、ユーザーからの要求をソフトウェアシステムに転換するために成果物の作成に関する一連の作業内容と作業内容の順序を定義しており、小さなリリースを繰り返しながら全体的なシステムを構築していく反復型アプローチを採用する。さらにモデリング言語として業界標準のUML(Unified Modeling Language) を採用する。従来のウォーターフォール型プロセスに比べるとRUP プロセスの特徴は、以下のものが挙げられる。

- ユースケース駆動　ユースケースはシステムの機能要求である。ユースケース駆動とは、開発チームが要求収集から実装まですべての開発作業をユースケース中心に進めていくことである。
- アーキテクチャ中心　RUP プロセスのアーキテクチャは、ソフトウェアシステムの最も重要な静的な側面と動的な側面を具体的に表現したものであり、六つのモデルビューで構成する。RUP プロセスでは、アーキテクチャをシステムが依存する基礎基盤としてとらえ、開発チームの中心に位置づける。
- 反復型開発　　ウォーターフール型プロセスの改良として、RUP プロセスは小さなリリースを繰り返しながら全体的なシステムを構築していくというアプローチをとる。結果として開発対象のシステムに対する拡張変更差分を生み出す。

アーキテクチャは反復的での作業を導く構造を提供し、ユースケースは目標を定義し、各反復の作業を決める。

1.2 RUP プロセスの問題点

開発現場の現状では、成果物を作成するために成果物の作成に関する作業内容と作業内容の順序に従うと同時に、重要なことについて他の誰かとコミュニケーションを行う必要もある。たとえば、システムの機能要求を再確認するために、管理者が顧客に適切な質問をしたり、成果物の設計における認識のズレを解消するために、プログラマ間の議論が起こることもある。したがって、成果物の作成においては、作業内容と作業内容の順序に加えて、開発者間のコミュニケーションの支援が重要である。

しかし、成果物を作成するために、RUP プロセスは作業内容と作業内容の順序を定義しているが、開発者間のコミュニケーションの支援についてはふれていない。

いくつかの開発プロセスとパターン言語は、開発時のコミュニケーションについて論じている。この中に、Coplien の組織パターン (Organizational Pattern) はソフトウェア開発をすすめる上で開発の進め方や組織構造のあり方についてコミュニケーションパスの構成法の観点から記述されたパターンである。

Coplien の組織パターンは、実際に何十という開発チームを調べ、その中で成功している開発チームの分析に基づき書かれており、パターン同士の多くが密接に関連しあうパターンランゲージの形をとっている。例えば、パターン 1 組織の大きさを決めよ（Size the Organization) は組織の大きさについて論じるとともに、組織の成長を支援するパターン 7 段階的実施（Phasing It In) との間に関連をつける。

したがって、RUP プロセスと Coplien の組織パターンを結合することによりコミュニケーション支援が可能なプロセスモデルを構築できる。

1.3 目的

本研究の目的は、RUP プロセスと Coplien の組織パターンを結合したプロセスモデルを定義し、プロセスモデルを用いて RUP における成果物中心の活動とコミュニケーションを融合する方式を提案することである。結果として成果物の作成において適切なコミュニケーションパスを示すことができる。ここの適切なコミュニケーションパスは、

- 取るべきコミュニケーションと取るべきではないコミュニケーション
- 作業の流れとは独立したコミュニケーション

を意味する。

RUP プロセスと Coplien の組織パターンを結合するために、Coplien の組織パターン構成の基本 (役割とコミュニケーション) を用いて、RUP の役割の管理モデルを作成し、Coplien による組織パターンの役割の管理モデルによる整理を行う。

1.4 論文の構成

本論文の構成は次の通りである。

- 第２章で Coplien の組織パターン構成の基本を述べ、RUP の役割の管理モデルを定義し、RUP プロセスと Coplien の組織パターンの結合を実現する。
- 第３章で RUP と Coplien の組織パターンを結合したプロセスモデルを定義する。
- 第４章で簡単な利用例を用いて本プロセスモデルの有効性を示す。
- 第５章でまとめと今後の課題を述べる。

第2章　RUPとCoplienの組織パターンの結合

本研究では、RUP プロセスと Coplien の組織パターンを結合したプロセスモデルを用いて RUP プロセスの成果物中心の活動とコミュニケーションを融合する方式を提案する。そのために RUP プロセスと Coplien の組織パターンの結合をどのように実現するのかが重要な点となる。この章では、まず Coplien の組織パターン構成の基本 (役割とコミュニケーション) を述べる。次に、RUP の役割の管理モデルを定義する。最後に、本研究に採用される Coplien の組織パターンを紹介し、Coplien の組織パターンの役割の管理モデルによる整理を述べる。

2.1　Coplienの組織パターン構成の基本

Coplien の組織パターンの多くは、開発組織の編成を各開発者に割り当てられた役割という定義で分析する。J.Coplien は、役割と責任範囲、コミュニケーションを以下のように分析している。これを図 2.1 で示す。

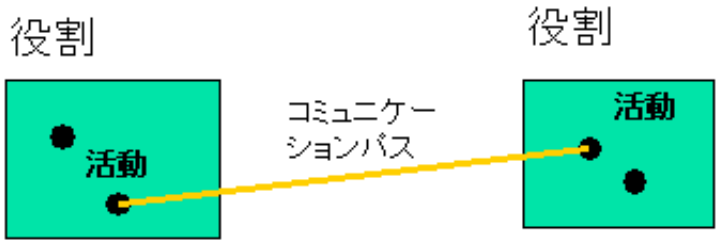


図 2.1: Coplien の組織パターン構成の基本的考え方

- パターン 5: 形式は機能に従う（Form Follows Function）:密接に関係するソフトウェアエンジニアリング活動（すなわち、実装においてお互いに結びつけられたり、同じ産物を扱ったり、また、意味上同じドメインに関連するようなアクティビティ）をグループ化せよ。グループ化した活動には名前を与え、役割とせよ。関連活動は、そうした役割の責任（仕事の説明）となる。

- パターン 26:流通範囲を整える（Shaping Circulation Realms）:役割間の適切なコミュニケーション構造は、組織の成功の秘密である。コミュニケーションは、たった一つの役割ではコントロールできず、少なくとも二つの役割が必要である。コミュニケーションは、アクティビティ間の意味的結合に従う。

2.2 RUP プロセスの役割の観点による整理

RUP プロセスは、一般的なプロセスの定義に当てはまる。つまり、顧客の要求をソフトウェアシステムに変換するために、開発チームが行うソフトウェアエンジニアリング活動（アクティビティ）の集合である。関連するアクティビティをまとめるために、RUP プロセスも役割の定義を使っており、RUP プロセスの役割オーバービューが存在する。

そこで、RUP プロセスにおける役割を調査し、RUP プロセスの役割の抽象を行う。図 2.2 は、RUP プロセスの役割を抽象したものである。

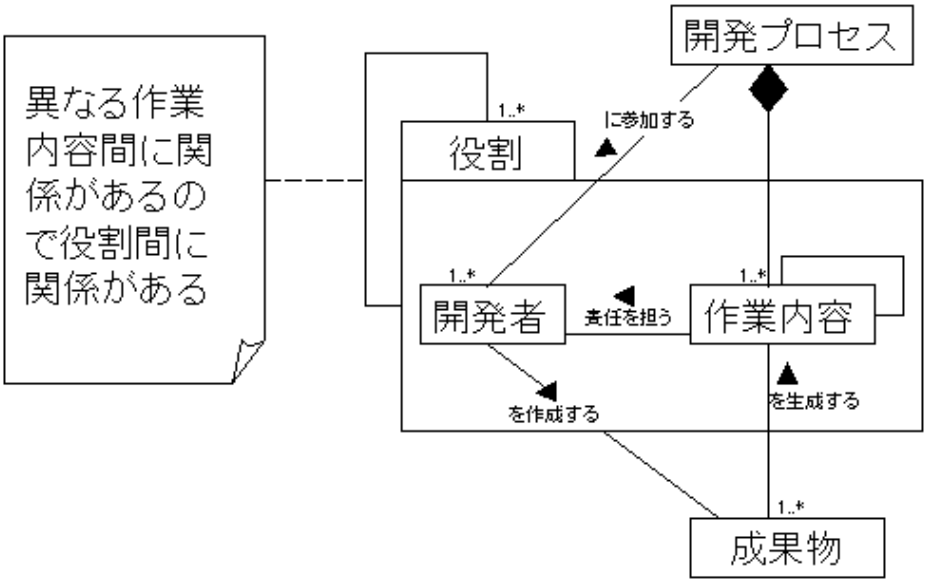


図 2.2: RUP の役割のメタモデル

RUP プロセスは、成果物を作成するために密接に関係するアクティビティをグループ化し、このグループを作業内容と定義する。例えば、ユースケースを設計するために、クラスを洗い出し、オブジェクトの相互作業を記述し、特殊要件を洗い出す。ゆえに、RUP プロセスは、この一連のアクティビティを、”ユースケースの設計 ”という作業内容として定義する。

RUP プロセスは、開発に参加する個人、サブ開発チームまたは開発チーム全体を開発者と定義する。作業内容は、開発者によって実行される。ゆえに、開発者と作業内容の間に責任関係がある。作業内容および作業内容に責任を担う開発者をパッケージ化して役

割を表現する。例えば、ユースケースエンジニアという役割を果たす開発者は、”ユースケースの設計 ”という作業内容の責任を担う。

各作業内容間に順序がある。例えば、”ユースケースの定義 ”という作業内容によって作成されたユースケース定義文にもとづいて、”ユースケースの分析 ”を行う。RUP プロセスは、作業内容間の順序を用いて、役割間の依存関係を表現する。

以上の一連の考察によって、RUP プロセスと Coplien の組織パターンが同じ角度（関連する活動の集合）から役割を定義している。RUP プロセスにおける役割の作業内容と Coplien による組織パターンの活動が類似すれば、結合を行うと考える。

2.3 RUP プロセスの役割の管理モデル

役割を用いて RUP プロセスと Coplien の組織パターンの結合を実現するために、RUP プロセスにおける様々な役割をまとめ、RUP の役割のメタモデルに基づき RUP の役割の管理モデルを定義する。

2.3.1 RUP プロセスの各役割

ここで、RUP プロセスにおける様々な役割をシステムエンジニア、アーキテクト、システム統合者、ユーザーインターフェイス、ユースケースエンジニア、コンポーネントエンジニアの６種類に分類する。役割の成果物および成果物に対する責任を用いて役割の作業内容を表現する。

- システムエンジニア

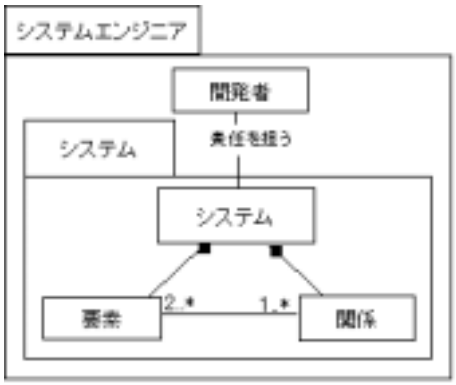


図 2.3: システムエンジニア

図 2.3 はシステムエンジニアを表す。システムエンジニアは、要求段階のシステム分析者とテスト段階のテスト設計者である。

要求段階のシステム分析者は、システムの境界を定め、アクターとユースケースを見つけだし、ユースケースモデルが完全で一貫性を持っていることを保証するという責任がある。一貫性を保つためには、要求を捕らえるときに、用語集を使って見解に達するようにする。

表 2.1: システム分析者の成果物

システム	要素	関係
ユースケースモデル	ユースケース アクター	ユースケースとアクター ユースケース間

テスト段階のテスト設計者は、テストモデルの整合性に対する責任を担い、モデルがその目的を果たすことを保証する。テスト設計者は、テストの適切な目標とテストのスケジュールを決め、テストケースおよび対応するテストプロシージャを記述するとともに、実行された統合テストとシステムテストを評価する責任も担う。テスト設計者が実際にテストを実行することはなく、むしろテストの準備と評価を中心に行う。

表 2.2: テスト設計者の成果物

システム	要素	関係
テストモデル	テストケース テストプロシージャ	テストケースとテストプロシージャ

- アーキテクト

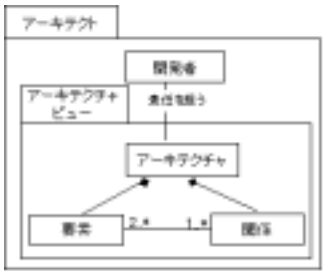


図 2.4: アーキテクト

図 2.4 はアーキテクトを表す。アーキテクトは、要求段階、分析段階、設計段階、実装段階、テスト段階のアーキテクトである。

要求段階にアーキテクトは、ユースケースモデルにおいて重要で不可欠なユースケースを記述し、反復を立案するためにユースケースに優先度をづけ、ユースケースモデルのアーキテクチャビューを記述する。

表 2.3: 要求段階のアーキテクトの成果物

アーキテクチャビュー	要素	関係
ユースケースモデルのアーキテクチャビュー	重要なユースケース	ユースケース間

分析段階にアーキテクトは分析モデルの整合性に対する責任を担い、全体としての分析モデルが正しく、一貫性があり、わかりやすいものであることを保証し、分析パッケージを洗い出し、アーキテクチャにとって重要な部分がモデルのアーキテクチャビューに描かれていることに対しても責任を担う。

表 2.4: 分析段階のアーキテクトの成果物

アーキテクチャビュー	要素	関係
分析モデルのアーキテクチャビュー	分析パッケージ ユースケース	分析パッケージとユースケース ユースケース間

設計段階にアーキテクトは設計モデルと配置モデルの整合性に対する責任を担い、全体としてのモデルが正しく、一貫性があり、わかりやすいものであることを保証し、サブシステムとノートを洗い出し、設計モデルと配置モデルのアーキテクチャにとって重要な部分がモデルのアーキテクチャビューに描かれていることに対しても責任を担う。

表 2.5: 設計段階のアーキテクトの成果物

アーキテクチャビュー	要素	関係
設計モデルのアーキテクチャビュー 配置モデルのアーキテクチャビュー	サブシステム インターフェイス ノート	サブシステム間 サブシステムとインターフェイス サブシステムとノート ノート間

実装段階にアーキテクトは実装モデルの整合性に対する責任を担い、全体としてのモデルが正しく、一貫性があり、わかりやすいものであることを保証し、重要な

コンポーネントを洗い出し、実装モデルのアーキテクチャにとって重要な部分がモデルのアーキテクチャビューに描かれていることに対しても責任を担う。さらに、実行可能コンポーネントをノートにマッピングすることに対しても責任を担う

表 2.6: 実装段階のアーキテクトの成果物

アーキテクチャビュー	要素	関係
実装モデルのアーキテクチャビュー 配置モデルのアーキテクチャビュー	コンポーネント インターフェイス ノート	コンポーネントとノート

- システム統合者

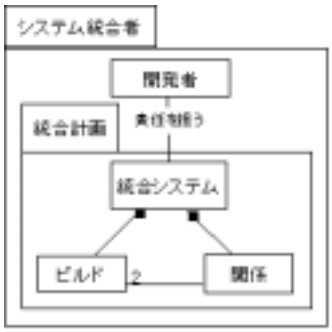


図 2.5: システム統合者

図 2.5 はシステム統合者を表す。システム統合者は、実装段階のシステム統合担当者である。

実装段階にシステム統合担当者の責任のなかには、各反復で作成する必要がある一連のビルドの計画と、各部分を実装したときのビルドの統合が含まれている。この計画の結果が統合化ビルド計画である。

表 2.7: システム統合者の成果物

要素	関係
ビルド	ビルド間

- ユーザーインターフェースデザイナー

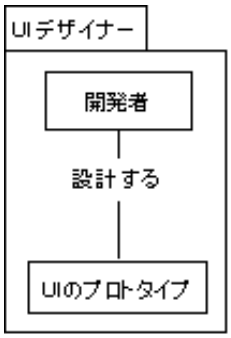


図 2.6: ユーザーインターフェースデザイナー

図 2.6 はユーザーインターフェースデザイナーを表す。ユーザーインターフェースデザイナーは、要求段階のユーザーインターフェースデザイナーである。

要求段階にユーザーインターフェースデザイナーは、ユーザーインターフェイスの外観をデザインし、ユーザーインターフェイスのプロトタイプを開発する。

- ユースケースエンジニア

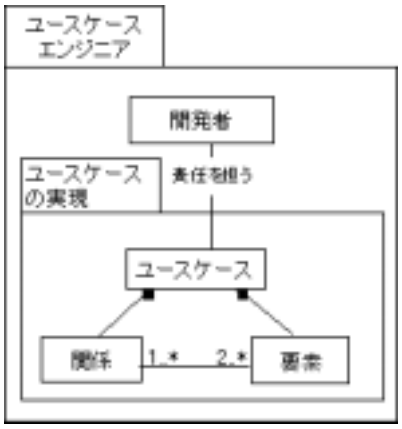


図 2.7: ユースケースエンジニア

図 2.7 はユースケースエンジニアを表す。ユースケースエンジニアは、要求段階のユースケース定義者と分析段階、設計段階のユースケースエンジニア、テスト段階のテスト担当者である。

要求段階にユースケース定義者は、ユースケースの詳細を記述する責任を持ち、担当するユースケースの実際のユーザーと緊密に作業を進める必要がある。

表 2.8: ユースケース定義者の成果物

ユースケースの実現	要素	関係
ユースケースの定義	-	-

分析段階にユースケースエンジニアは、一つまたは複数の ”ユースケースの実現ー分析 ”の整合性に対する責任を担い、分析クラスと分析オブジェクトの相互作用を洗い出し、”ユースケースの実現ー分析 ”が要求を満たしていることを確認する。” ユースケースの実現ー分析 ”は、ユースケースモデル中の対応するユースケースの振る舞いを正確に実現するものでなければならない。

表 2.9: 分析段階のユースケースエンジニアの成果物

ユースケースの実現	要素	関係
ユースケースの実現ー分析	分析クラス	分析クラス間

設計段階にユースケースエンジニアは、一つまたは複数の ”ユースケースの実現ー設計 ”の整合性に対する責任を担い、設計クラスと設計オブジェクトの相互作用を洗い出し、”ユースケースの実現ー設計 ”が要求を満たしていることを確認する。” ユースケースの実現ー設計 ”は、対応する ”ユースケースの実現ー分析 ”およびユースケースモデル中の対応するユースケースの振る舞いを正確に実現しなければならない。

表 2.10: 実装段階のユースケースエンジニアの成果物

ユースケースの実現	要素	関係
ユースケースの実現ー設計	設計クラス	設計クラス間
	インターフェイス	インターフェイスとサブシステム
	サブシステム	インターフェイスとサブシステム

テスト段階にテスト担当者は、作成された各ビルドに必要な統合テストを実行し、反復の結果全体を統合したビルドに必要なシステムテストを実行する責任を担う。テストの結果は、報告される欠陥である。

表 2.11: テスト担当者の成果物

ユースケースの実現	要素	関係
テストケース	欠陥	欠陥間

- コンポーネントエンジニア

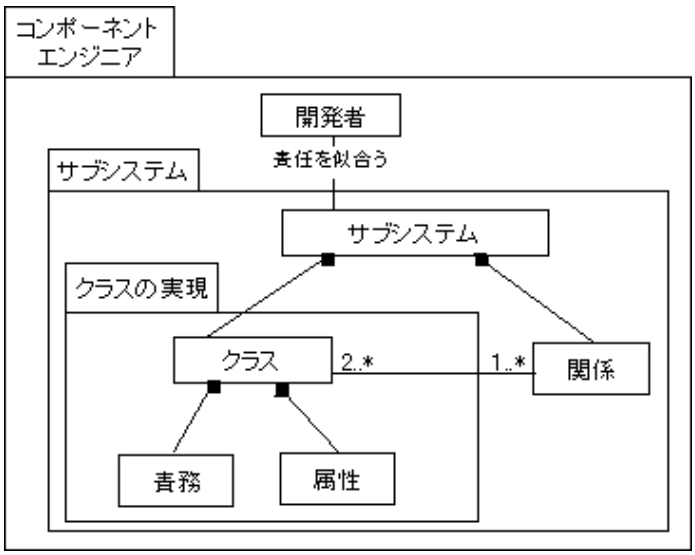


図 2.8: コンポーネントエンジニア

図 2.8 はコンポーネントエンジニアを表す。コンポーネントエンジニアは、分析段階、設計段階、実装段階のコンポーネントエンジニアである。

分析段階にコンポーネントエンジニアは、一つまたは複数の分析クラスの責務、属性、関係、特殊用件を定義して、維持し、分析クラスが参加している ”ユースケースの実現 ”で求められる要求を満たすことを確認する。

表 2.12: 分析段階のコンポーネントエンジニアの成果物

サブシステム	クラスの実現	属性	責務	関係
分析パッケージ	分析クラス	属性	責務	分析クラス間

設計段階にコンポーネントエンジニアは、一つまたは複数の設計クラスの操作、メソッド、属性、関係、実装要求を定義して維持し、設計クラスが参加している ”ユースケースの実現 ”で求められる要求を満たすことを確認する。

表 2.13: 要求段階のコンポーネントエンジニアの成果物

サブシステム	クラスの実現	属性	責務	関係
サブシステム	設計クラス	属性	操作	設計クラス間

実装段階にコンポーネントエンジニアは、一つまたは複数のファイルコンポーネントのソースコードを定義して保守し、各コンポーネントが正しく機能を実装していることを確認する。

表 2.14: 実装段階のコンポーネントエンジニアの成果物

サブシステム	クラスの実現	属性	責務	関係
サブシステム	コンポーネント	属性	操作	コンポーネント間

2.3.2 各役割間の関係の定義

各役割が作成する成果物間の関係を用いて、役割間の関係を表現する。各役割間の関係を実装関係、参照関係、トレース関係の 3 種類に分類する。

- 実現関係

  異なる役割の成果物に対して詳細設計または実装を行う場合、役割間に実現関係がある。

  1. 要求段階にユースケースエンジニアとシステムエンジニア間の関係は実現関係である。要求段階にユースケースエンジニアは、システムエンジニアが作成したユースケースモデルとそれに関連するユースケース図から始め、ユースケースを詳しく記述する。
  2. 分析段階にコンポーネントエンジニアとユースケースエンジニア間の関係は実現関係である。分析段階にコンポーネントエンジニアは、ユースケースエンジニアが作成した ”ユースケースの実現ー分析 ”に基づいて分析クラスの責務、属性、特殊用件を洗い出す。
  3. 分析段階にコンポーネントエンジニアとアーキテクト間の関係は実現関係である。分析段階にコンポーネントエンジニアは、アーキテクトが洗い出した分析パッケージがお互いにできる限り独立していることを確認する。
  4. 設計段階にコンポーネントエンジニアとユースケースエンジニア間の関係は実現関係である。設計段階にコンポーネントエンジニアは、ユースケースエンジ

ニアが作成した ”ユースケースの実現ー設計 ”に基づいて設計クラスの操作、属性、特殊要件を洗い出す。

5. 設計段階にコンポーネントエンジニアとアーキテクト間の関係は実現関係である。設計段階にコンポーネントエンジニアは、アーキテクトが洗い出したサブシステムがお互いにできる限り独立していることを確認する。
6. 設計段階にユースケースエンジニアとアーキテクト間の関係は実現関係である。設計段階にユースケースエンジニアは、”ユースケース実現ー設計 ”に必要なサブシステムの概要があるとき、その中に含まれるクラスのオブジェクトがサブシステムレベルでどのように相互作用するかを記述する。
7. 実装段階にコンポーネントエンジニアとシステム統合者間の関係は実現関係である。実装段階にコンポーネントエンジニアはサブシステムが統合ビルド計画に明記されているように、その役割を果たしているかを確認する。
8. テスト段階にテスト担当者とテスト設計者間の関係は実現関係である。テスト担当者は、テスト設計者が作成したテストケースのテストプロシージャを実行する。

- 参照関係

異なる役割の成果物を参照し、責任を果たすために新しい成果物を作成する場合、役割間に参照関係がある。

1. 要求段階にユーザーインターフェイスエンジニアとユースケース定義者、システム分析者間の関係は参照関係である。要求段階にユーザーインターフェイスデザイナーは、ユースケースモデルとユースケース詳細記述を出発点とし、ユーザーインターフェイスを設計する。
2. 要求段階にシステム分析者とアーキテクトの関係は参照関係である。要求段階にアーキテクトは、ユースケースモデルからユースケースモデルのアーキテクチャビューを取り込む。。
3. 要求段階のシステム分析者とテスト段階のテスト設計者の関係は参照関係である。テスト設計者は、ユースケース図とユースケースの記述文を参照に、テストケースとテストプロシージャをつくる。
4. 実装段階にシステム統合者とアーキテクトの関係は参照関係である。システム統合者は、ビルドを計画する。

- トレース関係

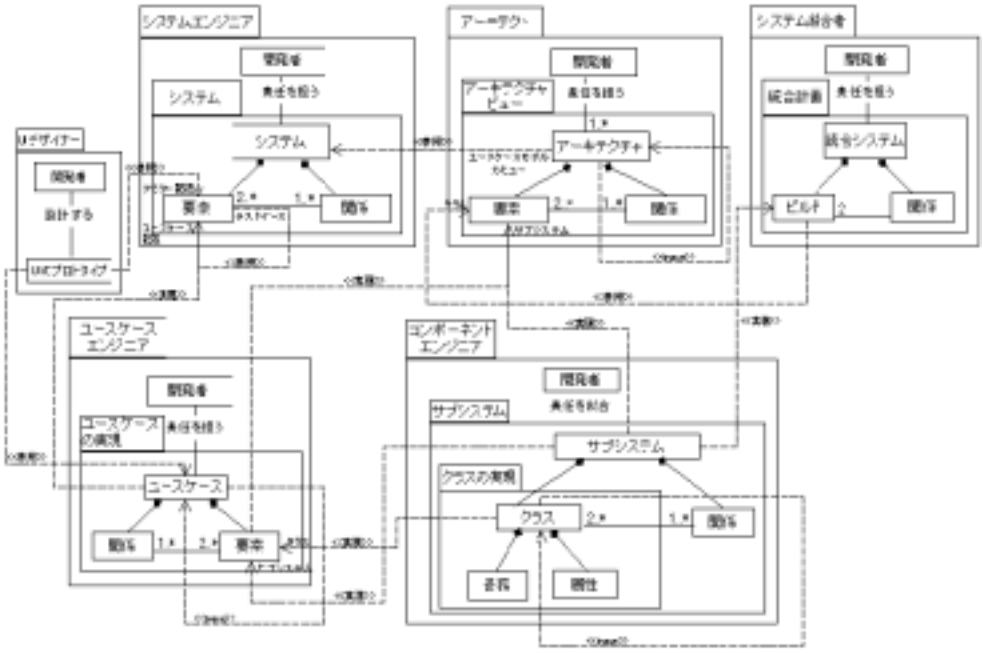
異なる開発段階の同一役割間にトレース関係がある。

1. 要求段階のユースケース定義者と分析段階のユースケースエンジニアの関係はトレース関係である。分析段階のユースケースエンジニアはユースケース定義をもとに、分析クラスを洗い出す。
2. 設計段階のユースケースエンジニアと分析段階のユースケースエンジニアの関係はトレース関係である。設計段階のユースケースエンジニアは ”ユースケー

ス実現ー分析 ”をもとに、設計クラスを洗い出す。

3. 設計段階のコンポーネントエンジニアと分析段階のコンポーネントエンジニアの関係はトレース関係である。設計段階のコンポーネントエンジニアは分析クラスの責務、属性をもとに、設計クラスの操作、属性を洗い出す。
4. 実装段階のコンポーネントエンジニアと設計段階のコンポーネントエンジニアの関係はトレース関係である。実装段階のコンポーネントエンジニアは設計クラスの操作、属性をもとに、クラスを実装する。

図を用いて、各役割間の実現関係、参照関係、トレース関係を表す。



2.4 役割についてのCoplienの組織パターン

Coplienの組織パターンには、役割に関わるパターンと役割に関わらないパターンの２種類がある。役割に基づいてRUPプロセスと結合するために、ここで役割に関わるCoplienの組織パターンに注目する。

- パターン11:開発者が工程をコントロールする (Developer Controls Process)

  開発プロセスの中枢に開発者 (Developer) の役割を置き、プロジェクトのコミュニケーションの中心は開発者にせよ。開発者は、工程の情報の集配器官だ。開発者の責任には、要求の理解、及び同僚とともに解決策の構造やアルゴリズムをレビューすること、実装の構築、そして単体テストが含まれる。開発者が主要な役割ととるべきだが、パターン24防火壁 (FireWall) との間でバランスをとるべきだ。

- パターン12:パトロン (Patron)

  このパターンは、パターン11:開発者が工程をコントロールするが実施されている場合にのみ機能する。プロジェクトの主義主張を擁護する、オープンで有能な管理者に、プロジェクトアクセス権を与えよ。パトロンは、プロジェクトで決定したことに最終的な判断を下す裁断者でもある。パトロンは組織に推進力を与え、意志決定を早める。パトロンには、進歩を妨げるプロジェクトレベルでの障壁を取り除く責任と、組織の志気を保つ責任がある。

- パターン13:アーキテクトが製品をコントロールする (Architect Control Product)

  プロジェクトにおけるパトロンの役割と同じ効果をアーキテクチャに対するアーキテクトの役割に期待する。アーキテクトの役割は、開発者の役割に助言をし、彼らを管理し、開発者たちと密にコミュニケーションをとるべきだ。アーキテクトはまた、顧客とも密接に関わり合うべきだ。

- パターン18:アプリケーションの設計はテスト設計に拘束される (Application Design Is Bounded by Test Design)

  シナリオ駆動型のテスト設計は、顧客によってシナリオの要求が最初に承認されたときに開始する。テスト設計は、ソフトウェア設計とともに進化するが、これは、顧客のシナリオの変化に応じる場合のみで、元となるソフトウェアは、テスト担当者は入手できない。

- パターン19:QAを引き込め (Engage QA)

  開発側で、何かテストすべきものができたら、すぐに開発とQAを、しっかりと結びつけよ。開発におけるテスト計画はコーディングと並行して進められるが、開発者は、システムにテスト準備が整ったことを最初に宣言しなければならない。QAの組織は、プロジェクトの外側にいるべきで、開発組織にテスト計画の説明をすべきではない。

- パターン 20:顧客を引き込め (Engage Customers)

  顧客の役割を、単に QA の役割だけではなく、開発者、およびアーキテクトの役割に密接に結びつけよ。

- パターン 24:防火壁 (FireWalls)

  プロジェクト外部からの干渉（批判など）から開発者を防ぐことを、プロジェクトマネージャの役割として定義せよ。

- パターン 25:門番（GateKeeper)

  技術者は、宿命的にひどくコミュニケーションが下手だ。だから、有効なコミュニケーション担当者が見つかれば、その能力を利用することが重要だ。プロジェクトの一員で、「タイプ E」の性格の持ち主を、門番の役割へ昇進させる。この人物は、プロジェクトの外側からの先端情報や、主流を逸脱した情報をプロジェクトのメンバーに広める。技術者は、GateKeeper を介してプロジェクト外部とのコミュニケーションをする。

2.5 Coplien の組織パターンの役割の管理モデルによる整理

前節までの分析に基づき Coplien の組織パターンと RUP プロセスの結合を行う。Coplien の組織パターンのコミュニケーションの定義を用いて RUP における各役割間のコミュニケーションパスを定義する。

2.5.1 RUP プロセスの役割と Coplien の組織パターン

Coplien による組織パターンの各役割と RUP プロセスの各役割を比較し、類似するアクティビティがあれば、結合を行う。ゆえに、以下の結合ができる。

- アーキテクトとパターン 12、パターン 13、パターン 20:

  RUP プロセスのアーキテクトは各開発段階のアーキテクチャビューに責任を担うので、Coplien の組織パターンのアーキテクトと共通のアクティビティがあると考える。ゆえに、RUP プロセスのアーキテクトプロジェクトとパターン 12、パターン 13、パターン 20 を結合する。結合した結果、プロジェクトにおけるパトロンの役割と同じ効果を、RUP プロセスのアーキテクトの役割に期待する。

- テスト設計者とパターン 18、パターン 19、パターン 20:

  ユースケースに基づき、RUP プロセスのテスト設計者はテストケースを設計するので、Coplien の組織パターンの QA と共通のアクティビティがあると考える。ゆえに、 RUP プロセスのテスト設計者とパターン 18、パターン 19、パターン 20 を結

合する。結合した結果、RUP プロセスのテスト設計者はテスト設計を行い、顧客との約束することで、品質保証の工程を開始できる。

- テスト担当者とパターン 19:

  RUP プロセスのテスト担当者は、テストケースを実行し、ビルドをテストするので、Coplien の組織パターンの QA と共通のアクティビティがあると考える。ゆえに、RUP プロセスのテスト担当者とパターン 19 を結合する。結合した結果、開発側で、何かテストすべきものができたら、すぐに開発と RUP プロセスのテスト担当者をしっかり結びつける。

- システム分析者とパターン 11、パターン 12、パターン 13、パターン 20、パターン 24、パターン 25:

  RUP プロセスのシステム分析者は、プロジェクトの外部にいる顧客のユースケースを洗い出すので、Coplien の組織パターンの開発者および門番と共通のアクティビティがあると考える。ゆえに、RUP プロセスのシステム分析者とパターン 11、パターン 12、パターン 13、パターン 20、パターン 24、パターン 25 を結合する。結合した結果、RUP プロセスのシステム分析者は、開発段階の責任を果たす。各開発者は、RUP プロセスのシステム分析者を介して顧客とコミュニケーションする。

- ユースケース定義者とパターン 11、パターン 12、パターン 13、パターン 20、パターン 24、パターン 25:

  RUP プロセスのユースケース転義者は、プロジェクトの外部にいる顧客のユースケースを詳細化するので、Coplien の組織パターンの開発者および門番と共通のアクティビティがあると考える。ゆえに、RUP プロセスのユースケース定義者とパターン 11、パターン 12、パターン 13、パターン 20、パターン 24、パターン 25 を結合する。結合した結果、RUP プロセスのユースケース定義者は、開発段階の責任を果たす。各開発者は、RUP プロセスのユースケース定義者を介して顧客とコミュニケーションする。

- ユースケースエンジニア (分析、設計) とパターン 11、パターン 12、パターン 13、パターン 24、パターン 25:

  RUP プロセスのユースケースエンジニア (分析、設計) は、ユースケースを分析するので、Coplien の組織パターンの開発者と共通のアクティビティがあると考える。ゆえに、RUP プロセスのユースケースエンジニア (分析、設計) とパターン 11、パターン 12、パターン 13、パターン 24、パターン 25 を結合する。結合した結果、開発組織において、RUP プロセスのユースケースエンジニア (分析、設計) は、開発者である。

- コンポーネントエンジニア (分析、設計、実装) とパターン 11、パターン 12、パターン 13、パターン 24、パターン 25:

RUP プロセスのコンポーネントエンジニア (分析、設計、実装) は、実装や単体テストを行うので、Coplien の組織パターンの開発者と共通のアクティビティがあると考える。ゆえに、RUP プロセスのコンポーネントエンジニア (分析、設計、実装) とパターン 11、パターン 12、パターン 13、パターン 24、パターン 25 を結合する。結合した結果、開発組織において、RUP プロセスのコンポーネントエンジニア (分析、設計、実装) は、開発者である。

- システム統合者 (実装) とパターン 11、パターン 12、パターン 13、パターン 19、パターン 24、パターン 25:

  RUP プロセスのシステム統合者 (実装) は、ビルドを統合するので、Coplien の組織パターンの開発者と共通のアクティビティがあると考える。ゆえに、RUP プロセスのシステム統合者 (実装) とパターン 11、パターン 12、パターン 13、パターン 19、パターン 24、パターン 25 を結合する。結合した結果、開発組織において、RUP プロセスのシステム統合者 (実装) は、開発者である。

組織パターン		RUPの役割
名前	説明	名前
Engage Customers	顧客の役割を開発者およびアーキテクトの役割に密接に結び付けよう	顧客 アーキテクト テスト設計者(テスト) システム分析者(要求) ユースケース定義者(要求)
Developer Controls Process	プロジェクトのコミュニケーションの中心は開発者にせよ。開発者の役割の責任には、要件の理解、解の構造やアルゴリズムのピュアレビュー、実装、単体テストなどがある	UIエンジニア(要求) ユースケースエンジニア(分析、設計 コンポーネントエンジニア(分析、設計、 システム統合者(実装) システム分析者(要求) ユースケース定義者(要求)
Patron	パトロンはプロジェクトで決定したことに最終的な判断を下す顧問者である。パトロンは組織に推進力を与え、意思決定を早める。パトロンには、進歩を妨げるプロジェクトレベルでの障壁を取り除く責任と組織の士気を保つ責任がある	プロジェクトマネージャ アーキテクト 顧客 開発者
Architect Controls Product	プロジェクトにおけるパトロンの役割と同じ効果を、アーキテクチャに対するアーキテクトの役割に期待する。アーキテクトの役割は、開発者の役割に助言をし、彼らを管理し、開発者たちと緊にコミュニケーションを取るべきだ。アーキテクトはまた、顧客とも密接に関わり合うべきだ。	アーキテクト 顧客 開発者
Firewalls	プロジェクト外部からの干渉(指示や入力等)から開発者を防ぐことを、プロジェクトマネージャの役割として定義せよ	プロジェクトマネージャ 顧客 開発者
GateKeeper	技術者は、指令的にコミュニケーションが下手だ。プロジェクト外部とのコミュニケーションはGateKeeperを介して行う。	システム分析者(要求) ユースケース定義者(要求) 顧客 開発者
Engage QA	QAを中心的な役割とせよ。開発側で、何かテストすべきものができたら、すぐに開発とQAをしっかりと結び付けよ。QAと顧客が協調することで、品質保証の工程(ユースケースを集めるなど)を開始できる	テスト設計者(テスト) テスト担当者(テスト) システム統合者(実装)
Application Design Is Bounded by Test Design	シナリオ駆動型のテスト設計は、顧客によってシナリオの要求が最初に承認された時に開始する。テスト設計はソフトウェア設計とともに進化するが、これは顧客のシナリオの変化に応じる場合のみで、元となるソフトウェアもテスト担当者は入手できない	テスト設計者(テスト) 顧客

図 2.9: Coplien による組織パターンと RUP の結合

2.5.2 役割の管理モデルにおける各役割間のコミュニケーションパス

Coplien の組織パターンは、コミュニケーションがアクティビティ間の意味的結合に従うと定義している。ゆえに、Coplien の組織パターンにおける各役割の間にコミュニケーションパスが存在すると考える。例えば、アーキテクチャレベルのパトロンは、アーキテクトである。開発者は、アーキテクチャを実現する。開発者間にアーキテクチャの設計に対する認識のズレが生じる場合、アーキテクトとのコミュニケーションを行い、アーキテクトが最後の結論をつける。

前節まで、RUP の役割の管理モデルにおける役割と Coplien の組織パターンを結合した。Coplien の組織パターンにおけるコミュニケーションの定義を用いて、RUP の役割の管理モデルにおける役割間のコミュニケーションパスを定義すると考える。

- システム分析者と顧客の間

  根拠となるパターンは、パターン 20:顧客を引き込め (Engage Customers)、パターン 25:門番 (GateKeeper) である。

  顧客満足度を達成するには、顧客の役割をアーキテクトや開発者の役割と関連づける。しかし、技術者は、宿命的にコミュニケーションが下手である。したがって、システム分析者を介して、プロジェクトの外側からの情報を開発者に広め、プロジェクトの情報を関連する部外者へリークする。

- ユースケース定義者と顧客の間

  根拠となるパターンは、パターン 20:顧客を引き込め (Engage Customers)、パターン 25:門番 (GateKeeper) である。

  顧客満足度を達成するには、顧客の役割をアーキテクトや開発者の役割と関連づける。しかし、技術者は、宿命的にコミュニケーションが下手である。したがって、ユースケース定義者を介して、プロジェクトの外側からの情報を開発者に広め、プロジェクトの情報を関連する部外者へリークする。

- システム分析者と開発者の間

  ここの開発者は、Coplien の組織パターンの役割であり、RUP プロセスのユースケースエンジニア、コンポーネントエンジニア、システム統合者、ユーザーインターフェイスである。

  根拠となるパターンは、パターン 20:顧客を引き込め (Engage Customers)、パターン 25:門番 (GateKeeper) である。

  顧客満足度を達成するには、顧客の役割を開発者の役割と関連づける。しかし、コミュニケーションのオーバヘッドは、外部協力者数とともに、上昇する。したがって、システム分析者を介して、開発者はプロジェクトの外側からの情報を取得し、開発の状況を関連する部外者へリークする。

- 顧客とプロジェクトマネージャの間

  根拠となるパターンは、パターン 20:顧客を引き込め (Engage Customers)、パターン 24:防火壁 (FireWall) である。

  顧客満足度を達成するには、顧客の役割を開発者の役割と関連づける。しかし、プロジェクトの開発者は、しばしば入力や批判を投げかける必要性を感じている部外者によって、悩まされる。割り込みの多くは雑音である。したがって、プロジェクトマネージャは、顧客とのやりとりから開発担当者を保護する。

- 開発者とプロジェクトマネージャの間

  ここの開発者は、Coplien の組織パターンの役割であり、RUP プロセスのユースケースエンジニア、コンポーネントエンジニア、システム統合者、ユーザーインターフェイスである。

  根拠となるパターンは、パターン 12:パトロン (Patron)、パターン 20:顧客を引き込め (Engage Customers)、パターン 24:防火壁 (FireWall) である。

  顧客満足度を達成するには、顧客の役割を開発者の役割と関連づける。しかし、プロジェクトの開発者は、しばしば入力や批判を投げかける必要性を感じている部外者によって、悩まされる。割り込みの多くは雑音である。したがって、プロジェクトマネージャは、顧客とのやりとりから開発担当者を保護する。

  プロジェクトマネージャにプロジェクトアクセス権を与える。プロジェクトマネージャは、プロジェクトで決定したことに最終的な判断を下す裁断者である。プロジェクトマネージャは開発に推進力を与え、意志決定を早める。

- アーキテクトとプロジェクトマネージャの間

  根拠となるパターンは、パターン 12:パトロン (Patron) である。

  プロジェクトマネージャにプロジェクトアクセス権を与える。プロジェクトマネージャは、プロジェクトで決定したことに最終的な判断を下す裁断者である。プロジェクトマネージャは開発に推進力を与え、意志決定を早める。

- アーキテクトと顧客の間

  根拠となるパターンは、パターン 13:アーキテクトは製品をコントロールする、パターン 20:顧客を引き込め（Engage Customers) である。

  プロジェクトにおけるパトロンの役割と同じ効果をアーキテクチャに対するアーキテクトの役割に期待する。アーキテクトは、アーキテクチャレベルの意志決定を早めるために、顧客と密接に関わり合う。

- アーキテクトと開発者の間

ここの開発者は、Coplien の組織パターンの役割であり、RUP プロセスのユースケースエンジニア、コンポーネントエンジニア、システム統合者、ユーザーインターフェイスである。

根拠となるパターンは、パターン 13:アーキテクトは製品をコントロールする

プロジェクトにおけるパトロンの役割と同じ効果をアーキテクチャに対するアーキテクトの役割に期待する。アーキテクトは、開発者の役割に助言をし、彼らを管理し、開発者たちと密にコミュニケーションをとる。

- テスト設計者と顧客の間

  根拠となるパターンは、パターン 18:アプリケーションの設計はテスト設計に拘束される (Application Design Is Bounded by Test Design), パターン 19:QA を引き込め (Engage QA) である。

  シナリオ駆動型のテスト設計は、顧客によってシナリオの要求が最初に承認されたときに開始する。QA と顧客が約束することで品質保証の工程（ユースケースを集めるなど）を開始できる。

- システム統合者とテスト設計者の間

  根拠となるパターンは、パターン 18:アプリケーションの設計はテスト設計に拘束される (Application Design Is Bounded by Test Design), パターン 19:QA を引き込め (Engage QA) である。

  開発側で、何かテストすべきものができたら、すぐに開発と QA をしっかりと結び付ける。テスト設計は、ソフトウェア設計とともに進化するが、これは、顧客のシナリオの変化に応じる場合のみで、もととなるソフトウェアはテスト担当者はにゅうしゅうできない。

- 役割の管理モデルにおける各役割の間

  根拠となるパターンは、パターン 26(Shaping Circulation Realms) である。

  コミュニケーションは、アクティビティ間の意味的結合に従う。従って、役割の管理モデルにおける各役割間の関係は、開発における仕事の依存関係であり、各役割間のコミュニケーションパスである。

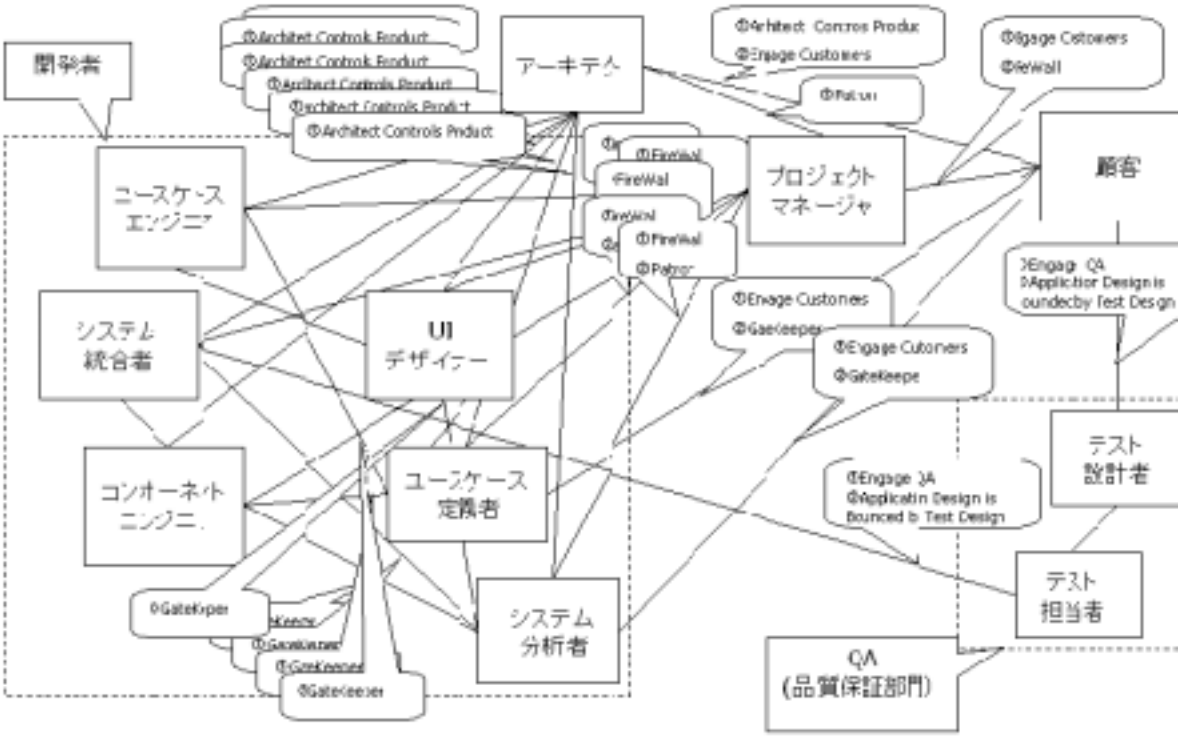


図 2.10: 役割間のコミュニケーションパス

第3章　RUPとCoplienの組織パターンを結合したプロセスモデル

第2章では、RUP の役割の管理モデルと Coplien の組織パターン構成の基本に基づき RUP における各役割間のコミュニケーションパスを定義し、成果物の作成において以下のようなコミュニケーションパスを示せるようになった。

- 取るべきコミュニケーションと取るべきでないコミュニケーション
- 作業の流れとは独立したコミュニケーション

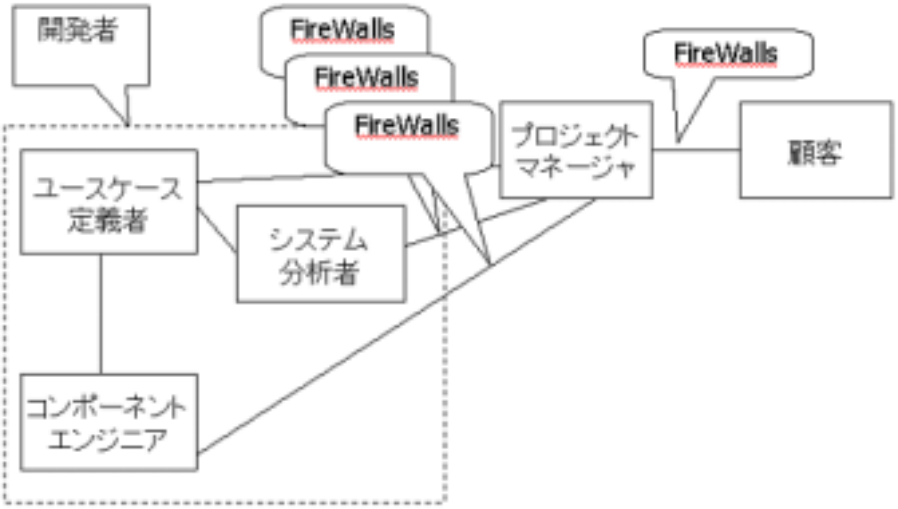


図 3.1: 防火壁　取るべきコミュニケーションと取るべきでないコミュニケーションを区別できる。

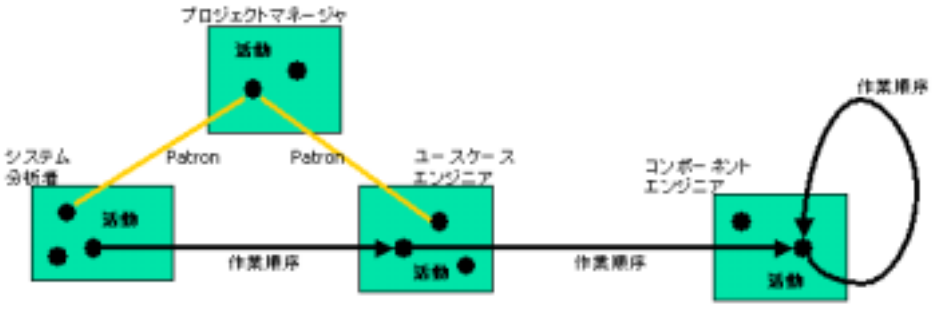


図 3.2: パトロン　作業の流れとは独立したコミュニケーションを示せる。

本章ではプロセスモデルにおける役割とコミュニケーションパスを再定義する。役割とコミュニケーションパスを再定義するために落水によるチーム構造モデル [1] を参照する。

3.1　役割オブジェクト

Coplien によると、「密接に関連する活動をグループ化し、役割と定義する」とある。図 2.9 における各役割が本ソフトウェアプロセスモデルの役割となる。役割の責任は、成果物関連責任とコミュニケーション責任に分類する。成果物に対する操作を用いて、役割の成果物関連責任を定義する。役割間のコミュニケーションパスを用いて、コミュニケーション責任を定義する。

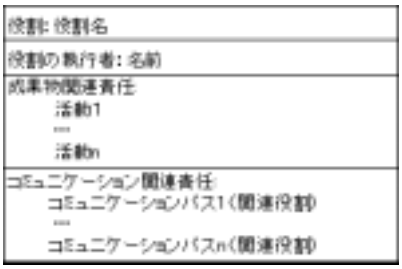


図 3.3: 役割

成果物関連責任とコミュニケーション責任の関係について、落水によると,”共同開発に参加する開発者は、コミュニケーションを通じて、共同開発のゴール、中間成果物やその要素の機能、実装法に関する共通認識に達する。共同開発のバックグランドとなる共通認識は、まず、状況の把握からはじまり、認識のズレの解消へと進む。コミュニケーションの結果、各自は新しい認識状態に達し、その状態を各自の責任範囲下にある中間成果物に反映させる、このような中間成果物の状態の変化は、再び、関係者間のコミュニケーションを引き起こす”とある。

各役割の成果物関連責任とコミュニケーション関連責任を定義する。

- システム分析者

  成果物関連責任
  1. 要求段階にアクターとユースケースを洗い出し、簡単に記述する
  2. 要求段階にユースケースモデルを完成する

  コミュニケーション責任
  1. システム分析者とユーザーインターフェイスデザイナー間
  2. システム分析者とアーキテクト間
  3. システム分析者とユースケースエンジニア間
  4. システム分析者とコンポーネントエンジニア間
  5. システム分析者とシステム統合者間
  6. システム分析者とテスト設計者間
  7. システム分析者とユースケース定義者間
  8. システム分析者とプロジェクトマネージャ間
  9. システム分析者と顧客間

- ユースケース定義者

  成果物関連責任
  1. 要求段階にユースケースを詳細に記述する

  コミュニケーション責任
  1. ユースケース定義者とユーザーインターフェイスデザイナー間
  2. ユースケース定義者とアーキテクト間
  3. ユースケース定義者とユースケースエンジニア間
  4. ユースケース定義者とコンポーネントエンジニア間
  5. ユースケース定義者とシステム統合者間
  6. ユースケース定義者とシステム分析者間
  7. ユースケース定義者とプロジェクトマネージャ間
  8. ユースケース定義者と顧客間

- アーキテクト

  成果物関連責任
  1. 要求段階にユースケースモデルのアーキテクチャビューの一部分として、重要不可欠なユースケースを記述する
  2. 要求段階にユースケースモデルのアーキテクチャビュの一部分として反復するためにユースケースの優先度を求める
  3. 分析段階に分析モデルのアーキテクチャビューの一部分として、分析パッケージを洗い出す

4. 分析段階に分析モデルのアーキテクチャビューの一部分そして、分析パッケージの依存関係を洗い出す
5. 設計段階に設計モデルのアーキテクチャビューの一部分として、サブシステムとそのインターフェイスを洗い出す
6. 設計段階に配置モデルのアーキテクチャビューの一部分として、ノードを洗い出し、ノードにサブシステムを分散する
7. 実装段階に実装モデルのアーキテクチャビューの一部分として重要なコンポーネントを洗い出す
8. 実装段階に配置モデルのアーキテクチャビューの一部分として、ノードにコンポーネントを分散する

コミュニケーション責任

1. アーキテクトとシステム分析者間
2. アーキテクトとシステム統合者間
3. アーキテクトとコンポーネントエンジニア間
4. アーキテクトとユースケースエンジニア間
5. 異なる開発段階のアーキテクト間
6. アーキテクトとユースケース定義者
7. アーキテクトとユーザーインターフェイスデザイナー間
8. アーキテクトとプロジェクトマネージャ間
9. アーキテクトと顧客間

- システム統合者

  成果物関連責任

  1. 実装段階にビルドの計画を立てる
  2. 実装段階にシステムを統合する
  3. 実装段階に各ビルドを統合する

  コミュニケーション責任

  1. システム統合者とアーキテクト間
  2. システム統合者とコンポーネントエンジニア間
  3. システム統合者とシステム分析者間
  4. システム統合者とプロジェクトマネージャ間
  5. システム統合者とテスト担当者間
  6. システム統合者とユースケース定義者

- ユーザーインターフェイスデザイナー

  成果物関連責任

  1. 要求段階にユーザーインターフェイスを設計する

  コミュニケーション責任

1. ユーザーインターフェイスデザイナーとシステム分析者間
2. ユーザーインターフェイスデザイナーとユースケース定義者間
3. ユーザーインターフェイスデザイナーとアーキテクト間
4. ユーザーインターフェイスデザイナーとプロジェクトマネージャ間

- ユースケースエンジニア

  成果物関連責任
  1. 分析段階に ”ユースケース実現ー分析 ”の分析クラスを洗い出す
  2. 分析段階に ”ユースケース実現ー分析 ”の分析オブジェクトの相互作用を洗い出す
  3. 分析段階に ”ユースケース実現ー分析 ”の特殊要件を洗い出す
  4. 設計段階に ”ユースケース実現ー設計 ”の設計クラスを洗い出す
  5. 設計段階に ”ユースケース実現ー設計 ”の設計オブジェクトの相互作用を洗い出す
  6. 設計段階に ”ユースケース実現ー設計 ”の特殊要件を洗い出す

  コミュニケーション責任
  1. ユースケースエンジニアとシステム分析者間
  2. ユースケースエンジニアとユースケース定義者間
  3. ユースケースエンジニアとアーキテクト間
  4. ユースケースエンジニアとプロジェクトマネージャ間
  5. ユースケースエンジニアとアーキテクト間
  6. ユースケースエンジニアとコンポーネントエンジニア間
  7. 異なる開発段階のユースケースエンジニア間

- コンポーネントエンジニア

  成果物関連責任
  1. 分析段階に分析クラスの責務、属性を洗い出す
  2. 分析段階に特殊要件を洗い出す
  3. 分析段階に分析パッケージがお互いに独立していることを確認する
  4. 設計段階に設計クラスの操作、属性を洗い出す
  5. 設計段階に設計クラスの状態を記述する
  6. 設計段階にサブシステムがお互いに独立していることを確認する
  7. 実装段階にクラスを実装する
  8. 実装段階にサブシステムがその役割を果たしているかを確認する
  9. 実装段階に単体テストを行う

  コミュニケーション責任
  1. コンポーネントエンジニアとユースケースエンジニア間
  2. コンポーネントエンジニアとシステム統合者

3. 異なる開発段階のコンポーネントエンジニア間
4. コンポーネントエンジニアとシステム分析者間
5. コンポーネントエンジニアとユースケース定義者間
6. コンポーネントエンジニアとプロジェクトマネージャ間
7. コンポーネントエンジニアとアーキテクト間

- テスト設計者

  成果物関連責任
  1. 要求段階にテスト方針を記述する
  2. 要求段階に統合テストケースを設計する
  3. 要求段階にテストプロシージャを提案する

  コミュニケーション責任
  1. テスト設計者とシステム分析者間
  2. テスト設計者とテスト担当者間
  3. テスト設計者と顧客間

- テスト担当者

  成果物関連責任
  1. テスト段階にテストを行う

  コミュニケーション責任
  1. テスト担当者とテスト設計者間
  2. テスト担当者とシステム統合者

- 顧客

  コミュニケーション責任
  1. 顧客とシステム分析者間
  2. 顧客とアーキテクト間
  3. 顧客とユースケース定義者間
  4. 顧客とテスト設計者間
  5. 顧客とプロジェクトマネージャ間

- プロジェクトマネージャ

  コミュニケーション責任
  1. プロジェクトマネージャと顧客間
  2. プロジェクトマネージャとユースケースエンジニア間
  3. プロジェクトマネージャとコンポーネントエンジニア間
  4. プロジェクトマネージャとシステム統合者間
  5. プロジェクトマネージャとアーキテクト間

6. プロジェクトマネージャとシステム分析者間
7. プロジェクトマネージャとユースケース定義者間
8. プロジェクトマネージャとユースケースインターフェイスエンジニア間

3.2 コミュニケーションパスオブジェクト

Coplien による、”コミュニケーションパスは役割における活動間の意味的結合に従って設計する”とある。コミュニケーションの概要は、役割間の関係を表す。

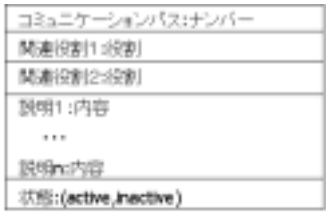


図 3.4: コミュニケーションパス

- システム分析者とアーキテクトの間

  1. 要求段階にアーキテクトは、システム分析者のユースケースモデルからユースケースモデルのアーキテクチャービューを作成する

  2. アーキテクトはアーキテクチャレベルの意志決定を早め、システム分析者に助言をし、システム分析者を管理し、密接にコミュニケーションをとる

- システム分析者とユースケース定義者の間

  1. 要求段階にユースケース定義者は、システム分析者のユースケースをもとにユースケースを定義する

  2. 要求段階にシステム分析者は、ユースケース定義者のユースケース定義をもとにユースケースモデルを完成する

- システム分析者とテスト設計者の間

  1. 要求段階にテスト設計者は、ユースケースモデルをもとにテストケースとテストプロシージャを作る

- システム分析者とユーザーインターフェイスデザイナーの間

  1. 要求段階にユーザーインターフェイスデザイナーは、ユースケースモデルをもとにユーザーインターフェイスを設計する

  2. システム分析者（要求）を介して、ユーザーインターフェイスデザイナーは、プロジェクト外部の顧客とコミュニケーションをする

- ユースケース定義者とユーザーインターフェイスデザイナーの間

  1. ユースケース定義者（要求）を介して、ユーザーインターフェイスデザイナーは、プロジェクト外部の顧客とコミュニケーションをする

  2. 要求段階にユーザーインターフェイスデザイナーは、ユースケース詳細設計をもとにユーザーインターフェイスを設計する

- アーキテクトとユースケース定義者の間

  1. アーキテクトはアーキテクチャレベルの意志決定を早め、ユースケース定義者に助言をし、ユースケース定義者を管理し、密接にコミュニケーションをとる

- アーキテクトとユーザーインターフェイスデザイナーの間

  1. アーキテクトはアーキテクチャレベルの意志決定を早め、ユーザーインターフェイスデザイナーに助言をし、ユーザーインターフェイスデザイナーを管理し、密接にコミュニケーションをとる

- 異なる開発段階のアーキテクトの間

  1. 分析段階にアーキテクトは、ユースケースモデルビューをもとに、分析パッケージを洗い出す

  2. 設計段階にアーキテクトは、分析モデルビューをもとに、サブシステム、インターフェイス、ノードを洗い出す

- アーキテクトとシステム統合者の間

  1. 実装段階にシステム統合者は、実装モデルと配置モデルをもとにビルドを計画する

  2. アーキテクトはアーキテクチャレベルの意志決定を早め、システム統合者に助言をし、システム統合者を管理し、密接にコミュニケーションする

- ユースケースエンジニアとアーキテクトの間

  1. 設計段階にユースケースエンジニアは、”ユースケース実現ー設計 ”に必要なサブシステムの概要があるとき、その中に含まれるクラスのオブジェクトがサブシステムレベルでどのように相互作用かを記述する

  2. アーキテクトはアーキテクチャレベルの意志決定を早め、ユースケースエンジニアに助言をし、ユースケースエンジニアを管理し、密接にコミュニケーションをとる

- ユースケースエンジニアとコンポーネントエンジニアの間

  1. 分析段階にコンポーネントエンジニアは、”ユースケース実現ー分析 ”をもとに、分析クラスの責務、属性、特殊要件を洗い出す

  2. 設計段階にコンポーネントエンジニアは、”ユースケース実現ー設計 ”をもとに、設計クラスの操作、属性、特殊要件を洗い出す

3. 設計段階にコンポーネントエンジニアは、サブシステムが適切なインターフェイスを提供していることを確認する

- 異なる開発段階のユースケースエンジニアの間

  1. 設計段階にユースケースエンジニアは、”ユースケース実現ー分析 ”をもとに、設計クラスを洗い出す

- ユースケースエンジニアとシステム分析者の間

  1. システム分析者（要求）を介して、ユースケースエンジニアは、プロジェクト外部の顧客とコミュニケーションをする

- ユースケースエンジニアとユースケース定義者の間

  1. 分析段階にユースケースエンジニアは、ユースケースの定義をもとに、分析クラスを洗い出す

  2. ユースケース定義者（要求）を介して、ユースケースエンジニアは、プロジェクト外部の顧客とコミュニケーションをする

- コンポーネントエンジニアとシステム分析者の間

  1. システム分析者（要求）を介して、コンポーネントエンジニアは、プロジェクト外部の顧客とコミュニケーションをする

- コンポーネントエンジニアとユースケース定義者の間

  1. ユースケース定義者（要求）を介して、コンポーネントエンジニアは、プロジェクト外部の顧客とコミュニケーションをする

- 異なる開発段階のコンポーネントエンジニアの間

  1. 設計段階にコンポーネントエンジニアは、分析クラスの責務、属性をもとに、設計クラスの操作、属性を洗い出す

  2. 実装段階にコンポーネントエンジニアは、設計クラスの操作、属性をもとに、クラスを実装する

- コンポーネントエンジニアとアーキテクトの間

  1. 分析段階にコンポーネントエンジニアは、分析パッケージがお互いにできる限り独立していることを確認する

  2. 設計段階にコンポーネントエンジニアは、サブシステムがお互いにできる限り独立していることを確認する

  3. アーキテクトはアーキテクチャレベルの意志決定を早め、コンポーネントエンジニアに助言をし、コンポーネントエンジニアを管理し、密接にコミュニケーションをとる

- コンポーネントエンジニアとシステム統合者の間

  1. 実装段階にコンポーネントエンジニアは、サブシステムが統合ビルド計画に明記されているように、その役割を果たしているかを確認する

  2. 実装段階にシステム統合者は、コンポーネントエンジニアが実装したコンポーネントを統合する

- システム統合者とシステム分析者の間

  1. システム分析者（要求）を介して、システム統合者は、プロジェクト外部の顧客とコミュニケーションをする

- システム統合者とユースケース定義者の間

  1. ユースケース定義者（要求）を介して、システム統合者は、プロジェクト外部の顧客とコミュニケーションをする

- テスト担当者とテスト設計者の間

  1. テスト段階にテスト担当者は、テストケースのテストプロシージャを実行する

- テスト担当者とシステム統合者の間

  1. システム統合者は、なにかテストすべきものができたら、すぐにテスト担当者にテストしてもらう

- 顧客とシステム分析者の間

  1. システム分析者（要求）を介して、コンポーネントエンジニアなどの開発者は、プロジェクト外部の顧客とコミュニケーションをとる

  2. アクターとユースケースを洗い出すために、システム分析者（要求）は、顧客からの情報提供が必要である

- 顧客とユースケース定義者の間

  1. ユースケース定義者（要求）を介して、コンポーネントエンジニアなどの開発者は、プロジェクト外部の顧客とコミュニケーションをとる

  2. ユースケース定義者（要求）は、自分が担当するユースケースの実際の顧客と緊密に作業を進める必要がある

- 顧客とプロジェクトマネージャの間

  1. プロジェクトマネージャは、決定したことに最終的な判断を下し、開発に推進力を与え、意志決定を早め、顧客と密接にコミュニケーションをとる

  2. プロジェクトマネージャは、プロジェクト外部の顧客からの干渉（批判など）からコンポーネントエンジニアなどの開発者を防ぐ

- 顧客とアーキテクトの間

  1. アーキテクトはアーキテクチャレベルの意志決定を早め、開発者に助言をし、顧客をアーキテクトに密接に結びつける

- 顧客とテスト設計者の間

  1. シナリオ駆動型のテスト設計者は、顧客によってシナリオの要求が最初に承認されたときに開始する

- プロジェクトマネージャとユースケースエンジニアの間

  1. プロジェクトマネージャは、プロジェクト外部の顧客からの干渉（批判など）からユースケースエンジニアを防ぐ

  2. プロジェクトマネージャは、決定したことに最終的な判断を下し、開発に推進力を与え、意志決定を早め、志気を保つ

- プロジェクトマネージャとコンポーネントエンジニアの間

  1. プロジェクトマネージャは、プロジェクト外部の顧客からの干渉（批判など）からコンポーネントエンジニアを防ぐ

  2. プロジェクトマネージャは、決定したことに最終的な判断を下し、開発に推進力を与え、意志決定を早め、志気を保つ

- プロジェクトマネージャとシステム統合者の間

  1. プロジェクトマネージャは、プロジェクト外部の顧客からの干渉（批判など）からシステム統合者を防ぐ

  2. プロジェクトマネージャは、決定したことに最終的な判断を下し、開発に推進力を与え、意志決定を早め、志気を保つ

- プロジェクトマネージャとシステム分析者の間

  1. プロジェクトマネージャは、プロジェクト外部の顧客からの干渉（批判など）からシステム分析者を防ぐ

  2. プロジェクトマネージャは、決定したことに最終的な判断を下し、開発に推進力を与え、意志決定を早め、志気を保つ

- プロジェクトマネージャとユースケース定義者の間

  1. プロジェクトマネージャは、プロジェクト外部の顧客からの干渉（批判など）からユースケース定義者を防ぐ

  2. プロジェクトマネージャは、決定したことに最終的な判断を下し、開発に推進力を与え、意志決定を早め、志気を保つ

- プロジェクトマネージャとユーザーインターフェイスエンジニアの間

  1. プロジェクトマネージャは、プロジェクト外部の顧客からの干渉（批判など）からユーザーインターフェイスエンジニアを防ぐ

  2. プロジェクトマネージャは、決定したことに最終的な判断を下し、開発に推進力を与え、意志決定を早め、志気を保つ

- プロジェクトマネージャとアーキテクトの間

  1. プロジェクトマネージャは、プロジェクト外部の顧客からの干渉（批判など）からアーキテクトを防ぐ

  2. プロジェクトマネージャは、決定したことに最終的な判断を下し、開発に推進力を与え、意志決定を早め、志気を保つ

第4章　有効性の確認

ここで ”UML による統一ソフトウェア開発プロセス ”[3] の具体例をプロセスモデルで再生成し、得られる結果を本に記述している結果と比較し、この手法の有効性を示す。

4.1　具体例

具体例として、”UML による統一ソフトウェア開発プロセス ”[3] に紹介した Interbank Consortium の例を用いた。対象とした開発段階は、要求段階と分析段階で、以下に各段階の役割と成果物を示す。役割のナンバーで成果物を生成する順序を示す。

- 要求段階
  1. 役割：システム分析者
     成果物：
       ユースケース：商品を注文する、注文を確認する、請求書の支払いをする、
                     購入者に請求書を送る
       アクター：購入者、販売者、会計システム
       ユースケースモデル
  2. 役割：アーキテクト
     成果物：
       ユースケースモデルのアーキテクチャビュー
  3. 役割：”請求書の支払いをする ”のユースケース定義者
     成果物：
       ”請求書の支払いをする ”ユースケースの詳細定義
  4. 役割：”購入者に請求書を送る ”ユースケース定義者
     成果物：
       ”購入者に請求書を送る ”ユースケースの詳細定義
  5. 役割：”請求書の支払いをする ”のユーザーインターフェイスデザイナー
     成果物：
       ”請求書の支払いをする ”ユーザーインターフェイス
  6. 役割：”購入者に請求書を送る ”のユーザーインターフェイスデザイナー
     成果物：
       ”購入者に請求書を送る ”ユーザーインターフェイス

- 分析段階
  1. 役割：アーキテクト
     成果物：
       サブシステム：購入者用請求書管理、購買者用請求書管理
  2. 役割：”請求書の支払いをする ”のユースケースエンジニア
     成果物：
       クラス：支払い要求ユーザーインターフェイス、注文ハンドラ、
               支払いスケジューラ、注文確認、請求書、支払い要求
  3. 役割：”請求書 ”クラスのコンポーネントエンジニア
     成果物：
       ”請求書 ”クラスの詳細設計

4.2 プロセスモデルの適用

プロセスモデルを用いて、役割間にコミュニケーションパスを付加した。その結果を図4.1で示す。

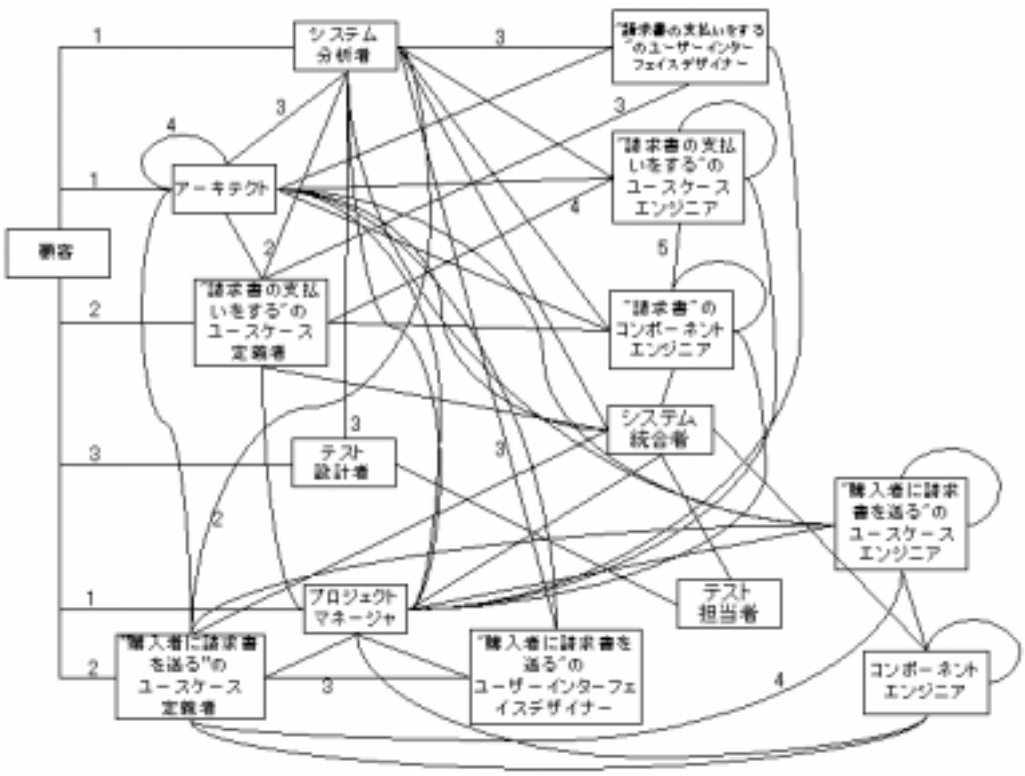


図 4.1: テストケース

各役割オブジェクト間のリンクは、コミュニケーションパスオブジェクトを示す。コミュニケーションパスオブジェクトに付加されるナンバーは、コミュニケーションパスオブジェクトがアクティブになる順序を示す。ナンバーが付加されないコミュニケーションパスオブジェクトは非アクティブ状態である。各コミュニケーションパスオブジェクトおよび役割オブジェクトについて説明する。

- 顧客とシステム分析者間

  アクターとユースケースを洗い出すために、システム分析者は、顧客からの情報提供が必要である。

  1. システム分析者
     アクター（購入者、販売者、会計システム）とユースケース（注文を確認する、請求書を支払いする、購入者に請求書を送る、商品を注文する）を洗い出し、簡単に記述する。
  2. 顧客

- 顧客とアーキテクト間

  アーキテクトは、アーキテクチャレベルの意志決定を早めるために、顧客と密接にコミュニケーションをする。

  1. アーキテクト
     ユースケースケースモデルのアーキテクチャビューの一部分として、反復するためにユースケースの優先度を求める。
  2. 顧客

- 顧客とプロジェクトマネージャ間

  決定したことに最終的な判断を下し、意志決定を早めるために、顧客と密接にコミュニケーションをする。

  1. プロジェクトマネージャ
  2. 顧客

- システム分析者と ”請求書の支払いをする ”ユースケース定義者間

  システム分析者が洗い出した ”請求書の支払いをする ”ユースケースをもとに ”請求書の支払いをする ”を定義する。システム分析者は、”請求書の支払いをする ”ユースケースの定義をもとに、ユースケースモデルを完成する。

  1. ”請求書の支払いをする ”ユースケース定義者
     ”請求書の支払いをする ”ユースケースを詳細に定義する。
  2. システム分析者
     アクター（購入者、販売者、会計システム）とユースケース（注文を確認する、請求書を支払いする、購入者に請求書を送る、商品を注文する）を洗い出し、簡単に記述する。

ユースケースモデルを完成する。

- 顧客と ”請求書の支払いをする ”ユースケース定義者間

  ”請求書の支払いをする ”ユースケース定義者は、担当する ”請求書の支払いをする ”ユースケースの実際の顧客と緊密に作業を進める。

  1. ”請求書の支払いをする ”ユースケース定義者
     ”請求書の支払いをする ”ユースケースを詳細に定義する。
  2. 顧客

- システム分析者と ”購入者に請求書を送る ”ユースケース定義者間

  システム分析者が洗い出した ”購入者に請求書を送る ”ユースケースをもとに ”購入者に請求書を送る ”を定義する。システム分析者は、”購入者に請求書を送る ”ユースケースの定義をもとに、ユースケースモデルを完成する。

  1. ”購入者に請求書を送る ”ユースケース定義者
     ”購入者に請求書を送る ”ユースケースを詳細に定義する。
  2. システム分析者
     アクター（購入者、販売者、会計システム）とユースケース（注文を確認する、請求書を支払いする、購入者に請求書を送る、商品を注文する）を洗い出し、簡単に記述する。
     ユースケースモデルを完成する。

- 顧客と ”購入者に請求書を送る ”ユースケース定義者間

  ”購入者に請求書を送る ”ユースケース定義者は、担当する ”購入者に請求書を送る ”ユースケースの実際の顧客と緊密に作業を進める。

  1. ”購入者に請求書を送る ”ユースケース定義者
     ”購入者に請求書を送る ”ユースケースを詳細に定義する。
  2. 顧客

- アーキテクトとシステム分析者間

  システム分析者のユースケースモデルからユースケースモデルのアーキテクチャビューを作成する。

  1. アーキテクト
     アーキテクチャビューの一部分として、重要不可欠なユースケースを記述する。
  2. システム分析者
     ユースケースモデルを完成する。

- テスト設計者とシステム分析者間

  ユースケースモデルをもとにテストケースとテストプロシージャを作成する。

1. テスト設計者
   テスト方針を記述する。
   統合テストケースを設計し、テストプロシージャを提案する。
2. システム分析者
   ユースケースモデルを完成する。

- テスト設計者と顧客間

  シナリオ駆動型のテスト設計者は、顧客によってシナリオが承認されたときにテスト設計を開始する。

  1. テスト設計者
     テスト方針を記述する。
     統合テストケースを設計し、テストプロシージャを提案する。
  2. 顧客

- システム分析者と ”請求書の支払いをする ”ユーザーインターフェイスデザイナー間

  ”請求書の支払いをする ”ユーザーインターフェイスデザイナーは、ユースケースモデルをもとに ”請求書の支払いをする ”ユーザーインターフェイスを設計する。

  1. システム分析者
     ユースケースモデルを完成する。
  2. ”請求書の支払いをする ”ユーザーインターフェイスデザイナー
     ”請求書の支払いをする ”ユーザーインターフェイスを設計する。

- システム分析者と ”購入者に請求書を送る ”ユーザーインターフェイスデザイナー間

  ”購入者に請求書を送る ”ユーザーインターフェイスデザイナーは、ユースケースモデルをもとに ”購入者に請求書を送る ”ユーザーインターフェイスを設計する。

  1. システム分析者
     ユースケースモデルを完成する。
  2. ”購入者に請求書を送る ”ユーザーインターフェイスデザイナー
     ”購入者に請求書を送る ”ユーザーインターフェイスを設計する。

- ”購入者に請求書を送る ”ユースケース定義者と ”購入者に請求書を送る ”ユーザーインターフェイスデザイナー間

  ”購入者に請求書を送る ”ユーザーインターフェイスデザイナーは、”購入者に請求書を送る ”ユースケース定義者の詳細設計をもとに、”購入者に請求書を送る ”ユーザーインターフェイスを設計する。

  1. ”購入者に請求書を送る ”ユースケース定義者
     ”購入者に請求書を送る ”ユースケースを詳細に記述する。
  2. ”購入者に請求書を送る ”ユーザーインターフェイスデザイナー
     ”購入者に請求書を送る ”ユーザーインターフェイスを設計する。

- ”請求書の支払いをする ”ユースケース定義者と ”請求書の支払いをする ”ユーザーインターフェイスデザイナー間

  ”請求書の支払いをする ”ユーザーインターフェイスデザイナーは、”請求書の支払いをする ”ユースケース定義者の詳細設計をもとに、”請求書の支払いをする ”ユーザーインターフェイスを設計する。

  1. ”請求書の支払いをする ”ユースケース定義者
     ”請求書の支払いをする ”ユースケースを詳細に記述する。
  2. ”請求書の支払いをする ”ユーザーインターフェイスデザイナー
     ”請求書の支払いをする ”ユーザーインターフェイスを設計する。

- 異なる開発段階のアーキテクト間

  アーキテクトは、ユースケースモデルビューをもとに、”購入者用請求書管理 ”サブシステムと ”販売者請求書管理 ”サブシステムを洗い出す。

  1. 要求段階のアーキテクト
     ユースケースモデルを完成する。
  2. 分析段階のアーキテクト
     ”購入者用請求書管理 ”サブシステムと ”販売者請求書管理 ”サブシステムを洗い出す。

- ”請求書の支払いをする ”ユースケース定義者と ”請求書の支払いをする ”ユースケースエンジニア間

  ”請求書の支払いをする ”ユースケース定義者をもとに、”請求書 ”分析クラスを洗い出す。

  1. ”請求書の支払いをする ”定義者
     ”請求書の支払いをする ”ユースケースを定義する。
  2. ”請求書の支払いをする ”ユースケースエンジニア
     ”請求書 ”分析クラスを洗い出す。

- ”購入者に請求書を送る ”ユースケース定義者と ”購入者に請求書を送る ”ユースケースエンジニア間

  ”購入者に請求書を送る ”ユースケース定義者をもとに、分析クラスを洗い出す。

  1. ”購入者に請求書を送る ”定義者
     ”購入者に請求書を送る ”ユースケースを定義する。
  2. ”購入者に請求書を送る ”ユースケースエンジニア
     分析クラスを洗い出す。

- ”請求書の支払いをする ”ユースケースエンジニアと ”請求書 ”コンポーネントエンジニア間

”請求書 ”コンポーネントエンジニアは、”請求書の支払いをする ”ユースケース実現をもとに、”請求書 ”分析クラスの責務、属性、特殊要件を洗い出す。

1. ”請求書の支払いをする ”ユースケースエンジニア
   ”請求書 ”分析クラスを洗い出す。
2. ”請求書 ”コンポーネントエンジニア
   ”請求書 ”分析クラスの責務、属性、特殊要件を洗い出す。

4.3 比較

プロセスモデルを用いてコミュニケーションで生成した成果物と、具体例で生成した成果物を比較して有用性を検証する。

コミュニケーションパスがアクティブになる順序を用いて、成果物を生成する順序を示すと、以下のようになる。

1. システム分析者と顧客のコミュニケーションでアクター（購入者、販売者、会計システム）とユースケース（注文を確認する、請求書の支払いをする、購入者に請求書を送る、商品を注文する）を洗い出す。
   アーキテクトと顧客のコミュニケーションで反復するためのユースケース優先度を決める。
   プロジェクトと顧客のコミュニケーションでプロジェクトの状況を把握する。
2. ”請求書の支払いをする ”ユースケース定義者とシステム分析者、顧客のコミュニケーションで、”請求書の支払いをする ”ユースケースを詳細設計する。
   ”購入者に請求書を送る ”ユースケース定義者とシステム分析者、顧客のコミュニケーションで、”購入者に請求書を送る ”ユースケースを詳細設計する。
3. アーキテクトとシステム分析者のコミュニケーションでユースケースモデルビューを作成する。
   テスト設計者とシステム分析者、顧客のコミュニケーションでテストケースを作成する。
   ”請求書の支払いをする ”ユーザーインターフェイスデザイナーとシステム分析者、”請求書の支払いをする ”ユースケース定義者のコミュニケーションで ”請求書の支払いをする ”ユーザーインターフェイスを作成する。
   ”購入者に請求書を送る ”ユーザーインターフェイスデザイナーとシステム分析者、”購入者に請求書を送る ”ユースケース定義者のコミュニケーションで ”購入者に請求書を送る ”ユーザーインターフェイスを作成する。
4. 分析段階のアーキテクトと要求段階のアーキテクトのコミュニケーションで ”購入者用請求管理 ”サブシステムと ”販売者請求書管理 ”サブシステムを洗い出す。
   ”請求書の支払いをする ”ユースケースエンジニアと ”請求書の支払いをする ”ユースケース定義者のコミュニケーションで ”請求書 ”クラスを洗い出す。

5. ”請求書の支払いをする ”ユースケースエンジニアと ”請求書の支払いをする ”コンポーネントエンジニアのコミュニケーションで ”請求書 ”クラスの属性、責任を洗い出す。

以上の成果物を生成する順序と具体例の成果物を生成する順序を比較すれば、ほとんど一致したが、以下の点が一致しなかった。

1. テスト設計
   具体例では、要求段階にテストケースを作成しないが、プロセスモデルでは、テスト設計者とシステム分析者のコミュニケーションでユースケースを洗い出すと同時に、テストケースを作成する。ゆえに、統合テストのテストケースは、ユースケースを反映しやすい。これは、組織パターンを導入する手法の利点と考えられる
2. プロジェクトマネージャ
   プロセスモデルでは、ユースケースを洗い出す時に、プロジェクトマネージャと顧客のコミュニケーションを行う。ゆえに、プロジェクトマネージャが早い段階でプロジェクトの状況を把握できる。これは、組織パターンを導入する手法の利点と考えられる。
3. アーキテクト
   プロセスモデルでは、ユースケースを洗い出す時に、アーキテクトと顧客のコミュニケーションを行う。ゆえに、アーキテクトが早い段階でアーキテクチャの状況を把握できる。これは、組織パターンを導入する手法の利点と考えられる。

4.4 評価

プロセスモデルを利用して、各役割間のコミュニケーションによって作成される成果物と、具体例に作成される成果物がほとんど一致した。さらに、プロセスモデルにおける役割の成果物関連責任とコミュニケーション関連責任によって、仕事依存関係によって生成されるコミュニケーションパスと組織構造によって生成されるコミュニケーションパスを表現できる。本研究の方向性の有効性は確認された。

第5章　おわりに

5.1　まとめ

本論文は、RUP プロセスと Coplien による組織パターンを結合したプロセスモデルを定義し、プロセスモデルを利用して成果物中心の活動とコミュニケーションを融合する方式を提案した。まず、Coplien による組織パターン構成の基本（役割、コミュニケーション）と RUP プロセスの役割に注目し、役割を用いて Coplien による組織パターンと RUP プロセスを結合するために、RUP プロセスの役割管理モデルを定義した。RUP プロセスの役割管理モデルを用いて RUP プロセスと Coplien による組織パターンを結合し、RUP プロセスにおける各役割間にコミュニケーションパスを付けた。これによって、成果物の作成において取るべきコミュニケーションと取るべきではないコミュニケーション、作業の流れとは独立したコミュニケーションを示せるようになった。つぎに、落水によるチーム構造モデルに基づきプロセスモデルにおける役割とコミュニケーションパスを再定義した。この手法を具体例に適用した結果、成果物を作成するときに必要となるコミュニケーションを表現できる。プロセスモデルはまだ不完全であるが、本研究の有効性が示された。

5.2　今後の課題

5.2.1　プロセスモデルに成果物オブジェクトの導入

コミュニケーションにおいて、誰と何を話すべきかという情報が重要である。定義したプロセスモデルには役割とコミュニケーションパスを詳細に定義しているが、成果物の詳細情報を表現する成果物オブジェクトを定義していない。コミュニケーションが発生したときにクラスの属性などの情報が必要である。ゆえに、クラスの属性や文章のバージョンなどの情報を抽象した成果物オブジェクトをプロセスモデルの中に導入する必要がある。

5.2.2　プロセスモデルの精度

Coplien の定義した役割と RUP で定義した役割の対応によって Coplien による組織パターンと RUP を結合した。Coplien の定義した役割と RUP で定義した役割の対応において自分なりの解釈をしたりするところがある。したがって、Coplien の定義した役割と RUP で定義した役割を再吟味する必要がある。

謝辞

本研究を行うにあたり、多大な御助言、ご指導を賜りました北陸先端科学技術大学院大学 落水 浩一郎 教授に深く感謝の意を表します。

本研究を進めるにあたり、 種々の有益な御助言をいただきました、 情報科学センター 藤枝 和宏 博士、落水研究室 藤田 充典 氏をはじめ、研究を行う上で非常に役立つご意見、ご感想を寄せていただいた落水研究室の皆さまに深く感謝いたします。

本論文の審査にあたり、本学 篠田 陽一 教授、片山 卓也 教授には、種々の有益な御助言をいただきました。深く感謝いたします。

最後に、学業だけでなく、私生活の面からもご支援してくださいました家族、友人に深く感謝いたします。

参考文献

[1] 落水 浩一郎：“ソフトウェア分散共同開発のためのチーム構造モデルの定義と調整支援への応用”、情報処理学会、ソフトウェア工学研究会、SE-131、2001.

[2] 落水 浩一郎：“分散共同ソフトウェア開発に対するソフトウェアプロセスモデルに関する基礎考察”、電子情報通信学会、信学技報、SS2000-48、200101.

[3] Ivar Jacobson, Grady Booch, James Rumbaugh：“UML による統一ソフトウェア開発プロセス”、翔泳社、2000.

[4] Kendall Scott：“入門統一プロセス”, ピアソン・エデュケーション, 2002.

[5] PLoPD Editor, 細谷 竜一, 中山 裕子監訳：“プログラムデザインのためのパターン言語” , ソフトバンク　パブリッシング, 2001.

[6] H.E Eriksson, Magnus Penker：“UML ガイドブック”、エスアイビー・アクセス, 1998.

[7] Graig Larman：“実践 UML”, ピアソン・エデュケーション, 1998.

[8] @IT フォーラム：http://www.atmarkit.co.jp/