# 第5章

# Heavyweightプロセスについて

## 成否を分ける「カスタマイズ」

### プロセスの分類

本章では，Agileの特徴をより明確にする意味で，Heavyweightプロセス（開発方法論）を簡単に紹介します．

Agileの提唱者たちは，作業の種類や成果物等が詳細に定義されている方法論を，「Heavyweightプロセス」と呼んでいます．これは，文字通り「重量級」のプロセスであるという意味です．

また，その一方で，「開発中の作業や成果物の順序を明確に定義することで，生産性を向上させると同時に，作業担当者の属人性を排除しよう」と考えるのが特徴の伝統的な開発方法論を，「予見型プロセス」と呼んでいます．これは，「開発過程はすべて事前に定義してしまい，作業を予見的に行う」という特徴に基づいた呼び方です．

そして，Heavyweightプロセスと予見型プロセスとをまとめて，「Agileの価値体系とは異なる方法論である」と指摘することもあるようです．しかし，この説明は少し極端かつ曖昧なので，ここで「伝統的プロセス」「Heavyweightプロセス」「予見型プロセス」の定義を簡単に整理しておきます．

現時点では，こうした方法論についての標準的な定義は存在しておらず，このあたりの定義は曖昧です．各名称は，ある種感覚的かつ便宜的に用いられています．実際のところ，伝統的プロセス=Heavyweightプロセス=予見型プロセスという分類は正確ではありません．

確かに，ウォーターフォールのような伝統的プロセスの多くは，手順，成果物などが明確に定義され，Heavyweightプロセスの性質を持っていることが少なくありません．そこで，Agile開発を推奨している人々は，一昔以前に発表され，現在までに多く利用されてきたプロセスを「伝統的」と呼び，Heavyweightプロセスと同義に用いていることが多いようです．

しかしながら，実際にはいわゆる伝統的プロセスの中にも，Heavyweightではないものが存在します．たとえば，初期のオブジェクト指向プロセスの1つであるCode/Yourdon法は，さほど詳細な手順，成果物などは述べられていません．

また，Heavyweightであっても，純粋な予見型ではないプロセスも存在します．たとえば，Rational Unified Processは，手順や作業等が詳細に定義されていながら，要求変更に対応することも考慮しています．

各方法論の分類について，表1に簡単にまとめてみました．ただし，明確な分類は困難なので，あくまで参考として考えてください．今後ますます開発方法論同士が相

| | 手順の詳細度 | 対象規模 | 対象分野 | 具体例 |
|---|---|---|---|---|
| **伝統的プロセス** | 作業の種類，成果物，担当ワーカが定義されているものからフェーズの説明とフェーズ内の大まかな作業が記述されているものまで幅がある． | 特に規模の指定はしていないものが多いが，中規模から大規模を対象にしているものが多い． | 汎用的なものから特定の分野（制御系，ITビジネスetc）と，方法論によってさまざま． | Jackson法，1980年代までのRAD方法論，Code/Yourdon法，OMTなど． |
| **Heavyweightプロセス** | 作業の種類，成果物，担当ワーカが定義されている． | 基本的に中・大規模． | 汎用的なものが多い． | RUP，Jackson法，Octopus，80年代までのRAD方法論[注1]など． |
| **予見型プロセス** | 作業の種類，成果物，担当ワーカの定義が明確に定義されている． | 基本的に中・大規模． | ITなどビジネス変化が激しい分野から特に分野を指定しないものまである．特に航空宇宙，防衛産業などミッションクリティカルな分野． | 伝統的プロセスの中で，明確な作業，成果物の定義がされているもの． |

表1 各種プロセスの分類

互に影響を受け，お互いの長所を取り入れるようになると，いずれはこうしたカテゴライズがあまり意味を持たなくなるかもしれません．

簡単に分類を示したところで，実際にHeavyweight開発方法論に分類される代表的な開発方法論を紹介していきたいと思います．

## Heavyweightプロセスとは？

Heavyweightプロセスとは，

- 作業担当者（ワーカ）
- 作業（アクティビティ）
- 作業間の明確な順序
- 各作業の開始条件と終了条件
- 開発ライフサイクル
- 各作業の成果物etc.

などが明確に定義されているプロセスを指します．上流工程から下流工程まで，詳細にプロセスが記述されています．中にはRUPのように，従来のソフトウェア開発のフェーズ範囲[注2]だけでなく，ビジネス構想やビジネス企画についてのフローを含むものもあります．

ソフトウェア開発の種類や特徴によって，開発手順の内容やフローは異なります．たとえば，ITビジネスのソフトウェアシステムと航空宇宙のソフトウェア開発では，システムの特徴や開発制約もかなり違うはずです．

そのため，どのような種類のソフトウェア開発にも適用できるようにするためには，非常に多くの作業を詳細に定義しなければならず，プロセス定義全体の規模はまさに「重量級」となります．また，汎用的なものにするためには，プロセス定義自体はある程度の抽象度を持たせた内容にする必要があります．汎用的である分，プロジェクトや開発対象ソフトウェアの種類・規模によっては不必要な作業定義も，プロセスには含まれます．

したがって，Heavyweightプロセスを採

注1）ここでのRAD方法論は，Jam Martinたちによる提唱を意識しています．RAD方法論は，数あるRAD開発の総称として用いられることが多いため，厳密な分類は困難です．

注2）要求定義からテストまで．

用する際には，テーラリング（カスタマイズ）が必要になります．

Agileプロセスが，最初から必要最小限の作業だけを定義しており，ユーザが適切な作業や順序を検討し，臨機応変に使いこなすことを要求しているのに対して，Heavyweightプロセスは，自分たちの開発プロジェクトに必要／不要な部分を検討しつつ，カスタマイズしなければなりません[注3]．

### Heavyweightプロセスの狙いと変化への対応

一般に，Heavyweightプロセスは，「目標としている結果を得るためには，作業を計画し，管理することが必要である．プロセスは管理可能であり，だからこそ，プロセスから生成される成果に対して有効に作用することができる」という価値体系モデルに基づいています．

このモデルでは，開発プロセスや管理プロセスは，プロセスの内部（作業，成果物，作業間のフロー，成果物間の依存関係）を詳細に定義し，管理することが可能であり，また必要であると考えています．これは，人間にはあらかじめ決められた作業フロー，成果物を計画に沿って実践し，成果を出す任務が課せられていることを意味します．その狙いは，作業と成果物を定めることによって，**常に同じ結果を得ることを保証**することと，開発担当者による作業のバラツキという**属人性を可能な限り排除**することです．

しかし，実際には，最近のHeavyweightプロセスは，ウォーターフォール型開発プロセスが主流だった頃とは異なり，変化への対応を考慮した性格を持つようになってきています．たとえば，Heavyweightプロセスの代表的存在であるRational Unified Process（以後RUP）は，Agile方法論と同じ価値をいくつか共有しているため，プロセスに柔軟な考え方を導入しています[注4]．

### Heavyweightプロセスの特徴とメリット

Heavyweyghtプロセスでは，フェーズごとに必要な作業，作業間の流れ，成果物，作業に携わる担当者（ワーカ）が，開発上流の作業から下流まで詳細に定義されています．したがって，優れた開発プロセスを適用したい現場のエンジニアには有用です．

最近のHeavyweightプロセスのメリットを，以下に列挙します．

- 各フェーズ中の作業1つ1つが明確に示されており，詳しく解説されている
- 作業間のフローが明確で，Best Practiceな作業の進め方が示されている
- 作業と成果物が対応づけられており，担当者の定義も明確になっている
- 開発ライフサイクルのタイミングによる作業の進め方の違いが定義されている
- 反復型開発，プロジェクト管理の方法が定義されている etc.

## 代表的なHeavyweightプロセス

ここからは，代表的なHeavyweightプロセスのいくつかを紹介していきます．

### Rational Unified Process

最も代表的なHeavyweightプロセスといえば，米国Rationalの「Rational Unified Process」が挙げられます．通常「RUP（ラップ）」と略されて呼ばれることが多いです．

注3）RUPを採用したプロジェクトにありがちな失敗として，プロジェクトマネージャが，RUPを一切カスタマイズせずに書いてあることを全部やるように指示してしまうことがあると，Alistair Cockburnは『Agile Software Development』（参考文献12）で指摘しています．

注4）ただし，「コミュニケーション」「コミュニティ」という要素に対する姿勢が，Agileとは異なっています（参考文献12）．


図1 日本ラショナルソフトウェアのWebサイト（http://www.rational.co.jp）

RUPの提唱元である米国Rationalは，早くからオブジェクト指向技術に着目し，ソフトウェア開発への普及に力を入れてきた企業です．オブジェクト指向のコンサルテーション，各種Case Toolベンダとして世界中に多くの顧客を抱え，常に新しい話題を提供してきたオブジェクト指向企業のリーディング・カンパニーであり，業界の巨人です（図1）．

このRUPは，日本でも何冊もの解説書が翻訳されており，容易に入手可能で，RUPを適用した開発事例も少なくありません．RUPの影響力は大きく，Heavyweightプロセスを中心に，他の多くのソフトウェア開発方法論がRUPをベースとした開発方法論やプロセスを提唱しています．

## ●RUPの全体像と特徴

RUPは，予見型開発方法論としての性格を持つ反面，反復型の開発プロセスを定義しており，ビジネスの変化や要求仕様の変更に対する柔軟な考え方を導入しています．

RUPは，Heavyweightプロセスと呼ばれるように，概要を紹介するだけでもかなりの量になります．ここでは，RUPの概要をイメージできる程度の説明にとどめますので，詳細については参考文献を参照してください．

## ●フェーズ

RUPの大きな特徴の1つに，開発ライフサイクル全体を4つのフェーズに分けている点が挙げられます．各フェーズは，さらにフェーズ内を1～数回に分けた反復開発（イテレーションと呼ぶ）を通じて作業が進められます（図2）．

この反復は，図2に示されているようにエンジニアリングの作業として，「ビジネス工学（エンジニアリング）」から「導入」までのワークフローを行うことを原則とします．ただし，フェーズや開発の状況において，特に作業の必要はない場合は省略することもありえます．

### 方向づけフェーズ

方向づけフェーズの目標は，プロジェクトの目標に関して，利害関係者全員の同意を確立することにあります．プロジェクトの開発範囲の定義，最終製品の検収条件，開発を行う際に考慮すべき諸事情，および最も重要な要求の優先度づけや制約を定義します．

具体的には，開発企画書の企画と取りまとめ，リスク管理，要員配置，プロジェクト計画に関する候補案を，コスト，スケジュール，収益性の間のトレードオフ等を勘案して取り決めます．

### 推敲フェーズ

推敲フェーズの目標は，システムのアーキテクチャのベースライン構築にあります．アーキテクチャは，最も重要な要求（システムのアーキテクチャに強い影響を与えるもの）およびリスク評価を通じて発展させ

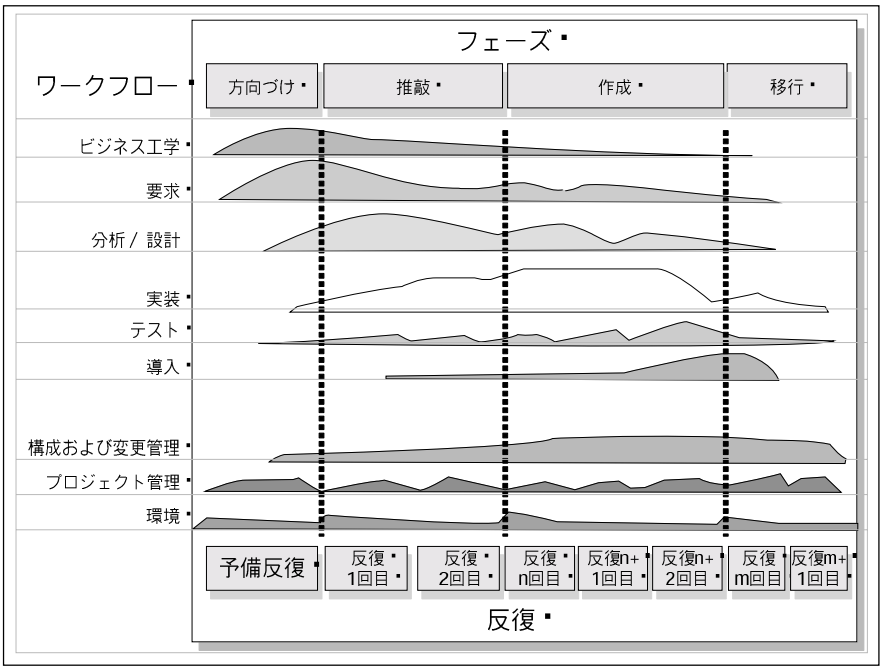



図2 RUPの全体の概要

ていく戦略をとっています．

このように，ソフトウェアアーキテクチャを重視して，反復型開発の中でアーキテクチャを評価しながら開発を進めていくアプローチを，**「アーキテクチャ中心開発」**と呼びます．アーキテクチャ中心開発は，RUPの中心的な戦略の1つとなっています．

推敲フェーズの狙いは，このソフトウェアアーキテクチャの検討と，その安定性を評価することにあります．このフェーズの終了のマイルストーンとして，次の2点を確認します．

- 開発完成までのコストとスケジュールを予測・決定するために，ソフトウェアアーキテクチャベースラインが確立されていること
- 要求および計画が十分安定しており，技術的なリスクも十分に軽減されていることの2点を確認すること

**作成フェーズ**

作成フェーズの目標は，未だ実現されていない残りの要求を，反復型開発を通じて実現していくことです．

技術的に難しいアーキテクチャの骨子は，推敲フェーズで検討され，実現および評価されています．作成フェーズは，反復開発を通じて，アーキテクチャに機能を評価しながら肉付けしていく作業フェーズになります．

ここでは開発者を増やし，サブシステムごとの並行開発を本格的に推進させていきます．そのため作成フェーズは，リソースの管理とコスト，スケジュールおよび品質を最適化するための業務管理に重点を置いた，製造のプロセスと考えることができます．

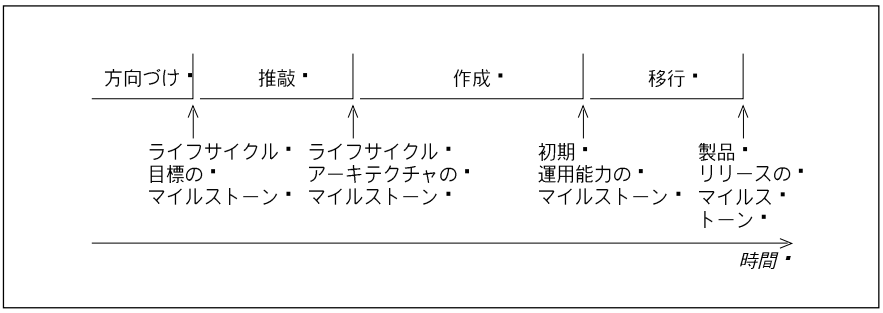



図3　フェーズとマイルストーンの関係

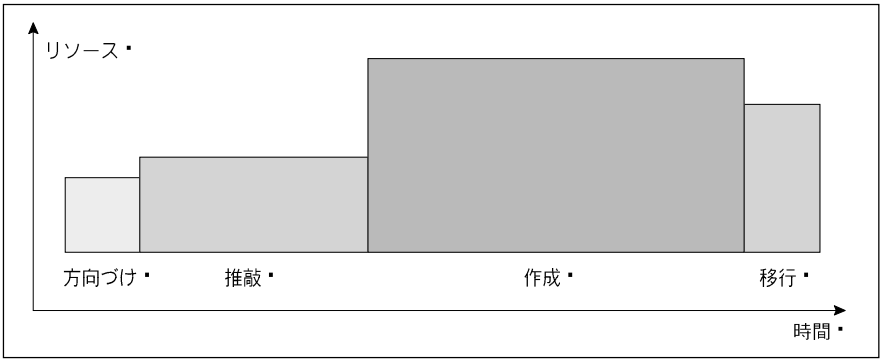



図4　各フェーズとリソースの関係

### 移行フェーズ

移行フェーズの焦点は，エンドユーザがソフトウェアを入手し，使用可能であることの確認にあります．複数の反復開発にわたる場合もあれば，1回だけの場合もあるというように，システムの特徴によってかなり異なります．

例としては，リリースの準備として製品をテストし，ユーザのフィードバックに基づいて，主に製品·カスタマイズインストール·使いやすさを微調整するといった作業が考えられます．

### フェーズの開始と終了

各フェーズは，基本的に2つの主要なマイルストーン間の期間で，マイルストーンの目的が達成された時点で終了となります（**図3**）．

各フェーズの終わりには，フェーズ開始時に立てた目標に対する成果を評価し，フェーズの目標が達成されたことを確認します．満足のいく評価を得た後は，プロジェクトを次のフェーズに進ませます．

各フェーズと作業量，担当者などのリソースとの関係は，**図4**のようになります．

### ワークフロー

ワークフローでは，一連の特定の成果物を作成するための作業（Activity）のすべてが表され，ウォーターフォール型開発のフェーズに対応しています．RUPでは，このように，ウォーターフォール型開発のフェーズは一回の繰り返しにマップされます．

さらにRUPには，「プロジェクト管理」

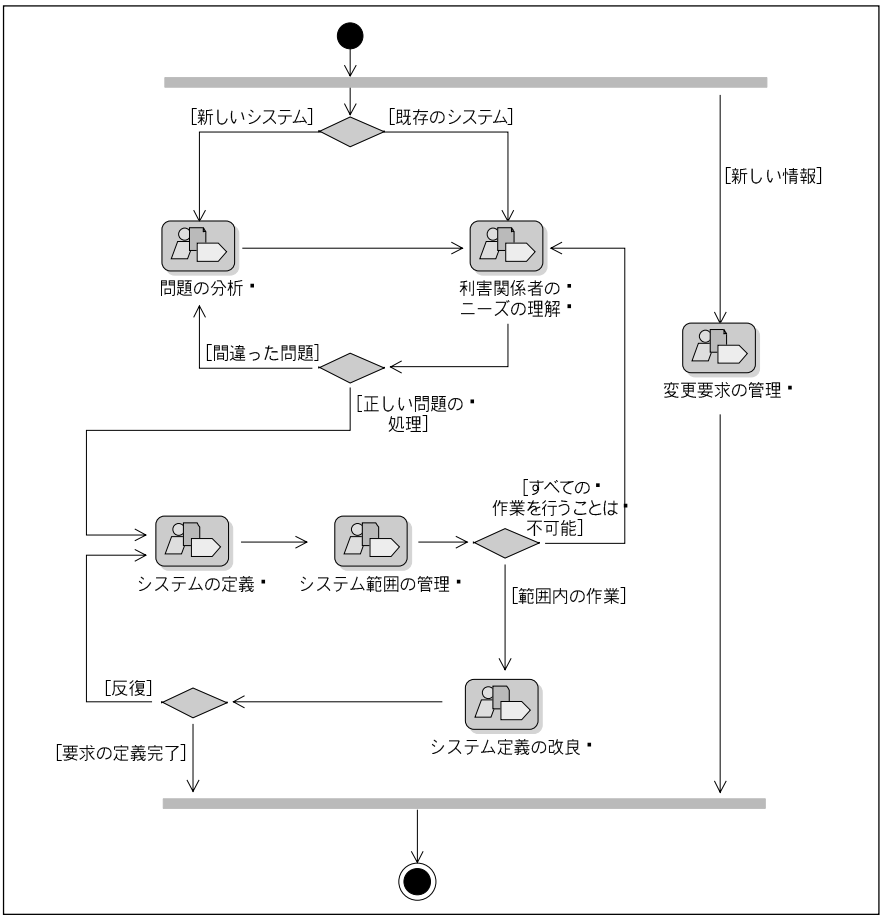


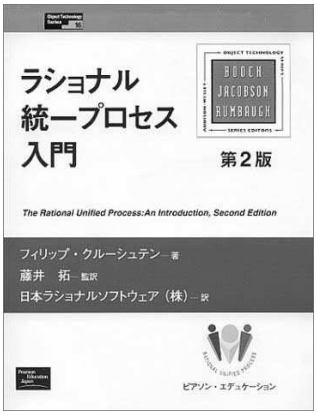

図5　要求分析のワークフローの例

「開発環境」「構成管理」など，開発業務に欠かせない作業である「プロジェクト管理プロセス」も盛り込まれており，エンジニアリング作業との対応が取られています．

また，反復型開発では，フェーズや反復によってワークフローの作業がいつも同じとは限らないため，図5のようなチャートで指針を示しています．

### アクティビティ，ワーカ，成果物

RUPでは，ワークフロー内はさらに「アクティビティ」と呼ばれる作業単位まで分解され，細かく定義されています．アクテ




RUPの入門書『ラショナル統一プロセス入門』（Philippe Kruchten著，藤井拓訳，ピアソン・エデュケーション，参考文献10）

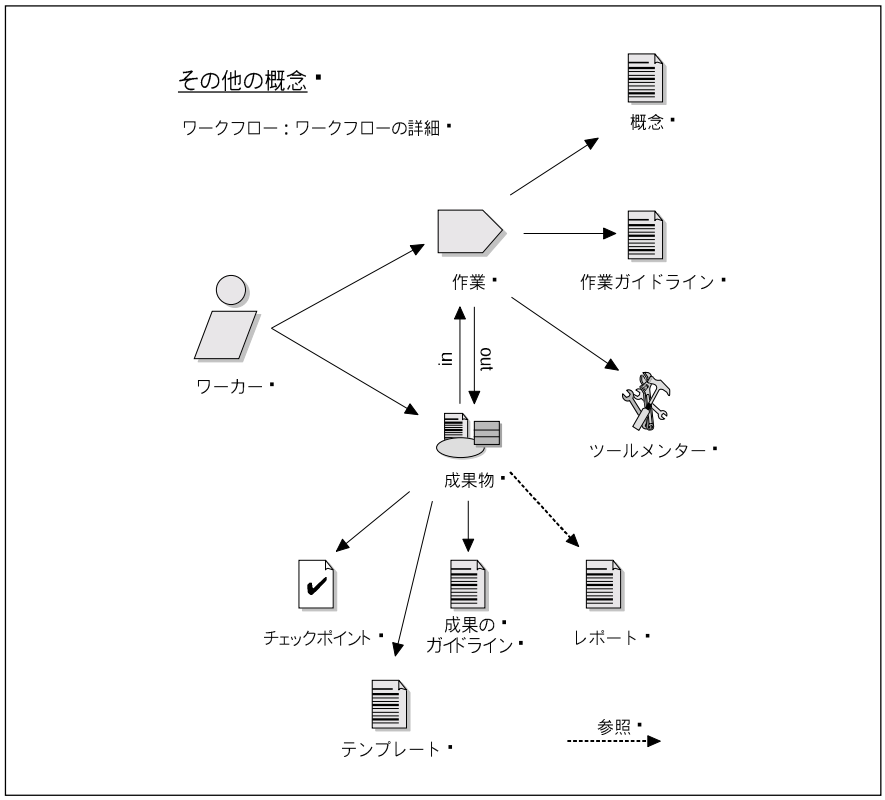



図6　RUPの用語とワーカ，成果物，アクティビティとの関係

ィビティとは，“これ以上は分解できない”作業単位のことです．

各アクティビティには，「ワーカ」と呼ばれる作業担当者の役割が明示されています．アクティビティには，作業する上で必要となる「入力成果物」と，作業結果である「出力成果物」が定義されています．

RUPの用語とワーカ，成果物，アクティビティとの関係を図6に，あるワークフローにおけるワーカ，成果物，アクティビティの関係を図7に，それぞれ示します．

## Information Engineering /RAD方法論

ここでは，Jam Martinたちの提唱によるIE（Information Engineering）方法論と，RAD（Rapid Application Development）方法論を紹介します．

この2つをHeavyweightプロセスとして分類すべきかについては，異論があるかもしれません．RAD方法論は，効果的に素早くソフトウェアを開発するという点において，むしろAgileに近いような印象も与えます．しかし，ソフトウェア開発で行うべき作業，必要となる成果物が非常に明確に定義されていることから，Heavyweightプロセスの章で紹介することにします．

◇　◇　◇

汎用的にすべてのソフトウェア分野を対象としているRUPとは異なり，IEとRADは，IT分野やデータを重視するシステムなどを中心に扱っており，ビジネスリエンジニアリングを成功させるシステム開発につ

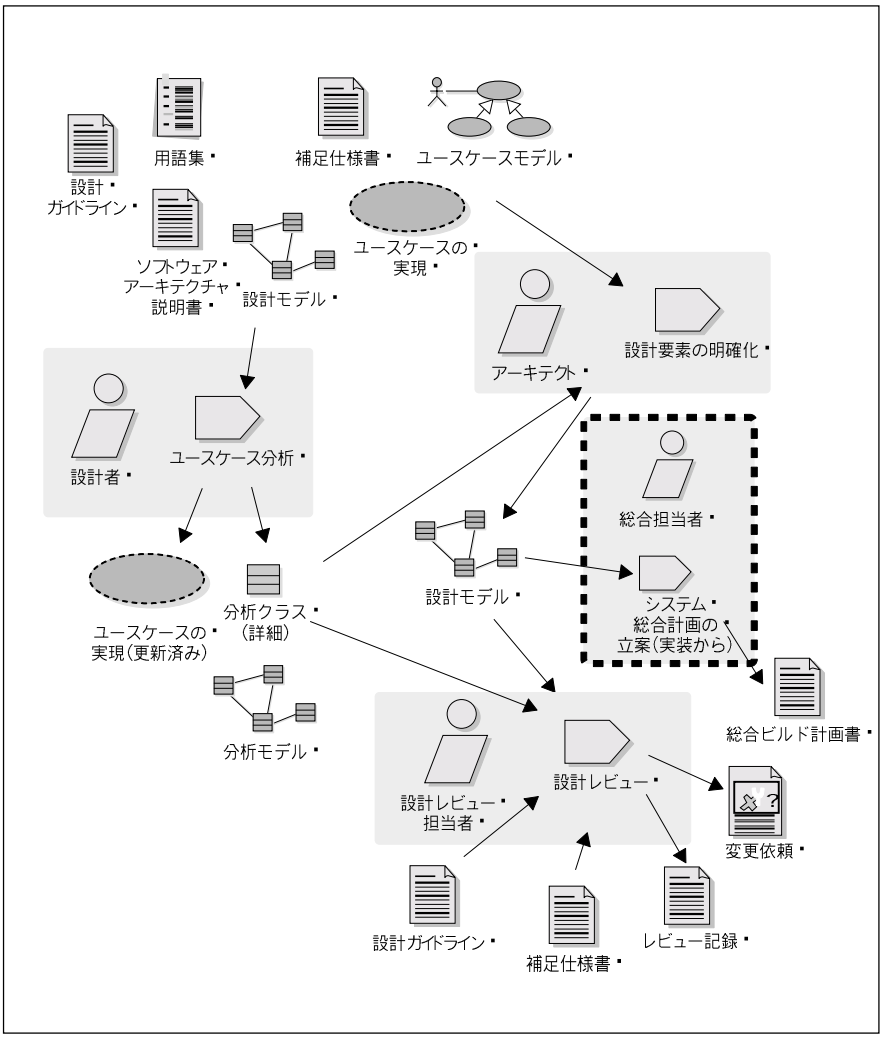



図7 あるワークフローにおけるワーカ，成果物，アクティビティの関係の例

いて解説しています．そして，ビジネスで成功するための要因として，"時間"を最も重要な価値の1つと位置づけています．

現在までに数多くのRAD方法論が発表され，開発に利用されています．そのため，一般に「RAD方法論」と呼ぶときには，特定の開発方法論の名前というよりも，"短期間に非常に効率的に開発を行う方法の総称"として用いられていることが多いです．ビジネス競争で優位に立つための要素として，"時間"が最重要のものの1つであることを重視した方法論です．ここで紹介するRAD方法論は，Jam Martinたちによる RAD方法であり，これまでに多くの短期開発方法論やRAD方法論に影響を与えてきたものです．

RAD方法論も，短期間に非常に効率的に開発を行う方法論ですが，最近のAgile方法論とはかなり異なっています．なお，ここで紹介するRAD方法論には詳しい解

説書があり，日本でも翻訳されて出版されています．ただし，原書の発行が10年以上も前で，内容的に若干古くなった部分があるのは否めません．しかし，その他の大部分は現在でも十分参考になり，役立つ内容となっています．

Jam Martinたちが提唱するRAD方法論は，IE方法論の後に発表されています．そのため，一般にはIE方法論の発展形として位置づけられているようです．

◇ ◇ ◇

IE方法論は，正確には単純に開発プロセスという性質のものではなく，ソフトウェア開発における企業戦略の重要性，戦略的システム構想，戦略的システム化計画などを含むBPR（Business Process Reengineering）**を推進する際の具体的な解説書**となっています．企業戦略の重要性，データモデリングの解説，BPR推進作業の指示が具体的かつ明確に述べられており，カスタマイズ可能なプロセス構成で，テンプレート化されています．

RAD方法論のほうは，IE方法論で紹介されている内容を，より開発プロセスモデルとして洗練させたものとなっています．

### ●Information Engineering／RADの特徴

前述したように，IE方法論やRAD方法論は，単なるソフトウェア開発の方法論ではなく，ソフトウェアによるBPRを意図している方法論です．

IT分野だけでなく，他のソフトウェア開発全般に共通する非常に重要な **“企業のビジネスゴールに直結したソフトウェア開発”** のポイントを，他の方法論よりも早期からプロセス中に具体的に盛り込んでいます．

さらに，非常に具体的で詳細なプロセスのテンプレートが掲載されており，1つ1つの作業（タスク）については，『インフォメーション・エンジニアリング』[注5]と『ラピッド・アプリケーション・デベロップメント』[注6]の中で詳しく解説されています．

この2冊は，単なる技術や方法論の啓蒙書ではなく，実用のための手順書として活用できるレベルまで具体的になっています．

内容は，企業戦略の重要性から会議室のレイアウト，開発者や管理者の精神的な話題[注7]まで多岐にわたっており，ソフトウェア開発プロジェクトの成功にはいろいろな要素が関わってくることに驚かされます．いくつかの例を以下に紹介します．

①プロジェクトを成功させるための，プロジェクトワーキンググループの構成についての考え方や作業の進め方
②ソフトウェア開発（主にデータベースシステム）に必要となる技術解説
③Meetingを行う部屋のレイアウト構成と議論の進め方
④ストレスなど，管理者のメンタルな部分への注意点
⑤Case Toolの選択と利用 etc

## Octopus

携帯電話で有名なフィンランドのNokiaによる，組込み・リアルタイムシステム用のオブジェクト指向開発方法論です（図10）．

近年の大規模化・複雑化するシステム開発に対応している，反復型の開発プロセスです．上流から下流まで，作業順序や成果物が非常に明確化しており，組込み・リアルタイムシステム特有の問題であるマルチ

注5）『インフォメーション・エンジニアリング』（James Martin著，三菱CC研究会IEタスクフォース訳，プレンティスホール／トッパン，参考文献7

注6）『ラピッド・アプリケーション・デベロップメント』（James Martin著，芦沢真佐子他訳，リックテレコム，参考文献6

注7）たとえば，RAD開発は厳しいタイムスケジュールで行われるため，“燃え尽き症候群”や開発者のモチベーションに注意を払うことが重要と述べられています．

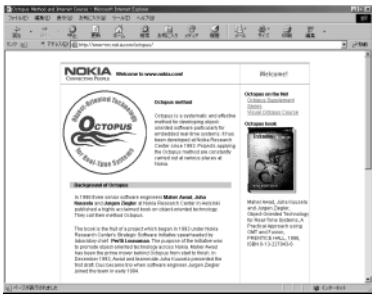



図10 NokiaによるOctopusのWebサイト
（http://www-nrc.nokia.com/octopus/）


図11 ROPESの提供元I-LogixのWebサイト
（http://www.ilogix.com/）

タスク構成でのタスク抽出や優先度づけの作業に対し，有効なプロセスを提供しています．

特に，プログラムの実行単位であるタスクとオブジェクトのマッピングについて，ユニークで効果的な方法を提唱しています．

## ROPES

USのI-Logix（図11）による組込み・リアルタイムシステム用のオブジェクト指向開発方法論です．Octopus同様，繰り返し型の開発プロセスであり，大規模・複雑化するシステム開発に対応しています．

ROPESでは，特に組込み・リアルタイムシステムで重要なオブジェクト状態の分析設計技法や，組込み・リアルタイムシステムに有効なパターンについて詳細に述べられています．

また，マルチタスク構成でのタスク抽出や優先度づけに関しては，数学的なアルゴリズムを積極的に利用するアプローチを取っています． ■