Yモデル型開発計画とピアレビューによる短工期化への対応

第２事業部
プロジェクト　プロフェッショナル

増子　博

Hiroshi　Masuko
hiroshi-masuko@exa-corp.co.jp

システム開発の短工期化と品質向上は開発方法論の古くからのテーマであり、現在に至るまで多くの方法論が発表され、今後もＩＴ技術の発展に沿って様々な提唱がされるであろう。実際のプロジェクトで開発計画を策定する場合に採用される方法論は、最も定番的なウォーターフォール（以下ＷＦ）型がその直感的な分かりやすさやステークホルダの経験や慣れから今後も主流を占めると考える。ＷＦ型の大きな問題として短工期化に対応しづらい点がある。近年、短工期システム開発の解決策の一つとしてさまざまな開発方法論（アジャイル開発方法論と呼ばれる）が提唱されている。しかし、それらの方法論は主に中小規模開発で本番リリースが段階的なケースで効果が発揮されるため適用できるプロジェクトが限定される。本論文ではＷＦ型での品質検証の基本的な考え方であるＶモデルを改良したＹモデルを提案し、また、アジャイル開発の実践や作法、特にピアレビューのアイデアをＹモデルと組み合わせた短工期化の解決策を提案している。

1. はじめに

システム開発の短工期化とコスト削減の要求は年々厳しさを増しており、プロジェクトマネージャは一層の生産性向上と利益確保の追求を常に求められている。システムを『速く(補足１)正確に』作るための短工期化と低コスト化の方法論が学会、書籍、公開サイトなどで数多く紹介されている。これらの方法論はどちらかというとフレームワークや共通ライブラリーの活用といった実装工程の生産性向上をテーマにしているものが多い。つまり『速く作る』解決策ではあるが『正確に作る』視点に欠けるものが多い。2001年のアジャイルアライアンス宣言（参考：http://agilemanifesto.org/)を契機に、システムを『俊敏に(補足１）正確に』作る方法論としていくつかの中軽量級開発方法論が注目された。これらの方法論が提唱する開発プロセスの長所・短所を検討し筆者の経験を加えて、ウォーターフォール型（以下"ＷＦ型"と記述）であるＶモデルソフトウェア開発（以下"Ｖモデル型"と記述）を応用した開発計画（以下、"Ｙモデル型"と記述）とピアレビュー（peer review）を組み合わせた実践的開発方法論をまとめた。本論文では、この方法論の提案と、期間６ケ月前後で開発量が中規模程度以下のプロジェクトへの適用を想定したいくつかの留意点や課題について説明する。

補足１：一般に短工期化で言う"速く"は要件定義からサービスインまでの時間の短縮を指すが、アジャイル開発で言う"俊敏"は、エンドユーザが感じる要求の実現度合い（＝顧客満足度）を分析しその結果を次のプログラムリリースに素早く反映することで結果的に短工期に対応することを指す。

---

2. 短工期開発の背景

2.1. なぜ、当初計画が遅延するのか

システム開発をＷＦ型の上流工程（本論文では、要求定義、外部設計、内部設計を指す）と下流工程（本論分では、製作/単体テスト、結合テスト、システムテスト、受け入れテストを指す）に分けて考えるとプロジェクトが遅延する典型的な原因は上流工程と下流工程で異なる。上流工程クとなることが多く、下流工程では"作業の手戻り"が原因になることが多い。下流工程での手戻りを引き起こす原因の大半が要求定義や外部設計の段階で発生すると言われている。これらに関して筆者自身は具体的なデータを持っていないが、経験的には参考文献５(17ページ)に記載されている『プロジェクトコストの30%から50%が手戻りのコストで、手戻りの70%から85%が要求の間違い』が一般的であると考える。プロジェクトの上流工程で遅延が発覚した場合はさまざまな対策を講じることが可能でありプロジェクトの失敗に直結することは少ないが、下流工程になるほど対応策の選択肢が狭まりリカバリーが難しい。したがって、プロジェクト下流工程での手戻りの発生をどれだけ抑止できるかが開発者側から見たプロジェクト成功の大きな要因になる。

2.2. 短工期に対応するための開発方法論

短工期開発に対応するためWF型をベースにしたプロジェクト計画でさまざまな工夫が長年行われてきた。しかし、WF型では（作業の戻りを認めない）フェーズドアプローチの特性などの制約で短工期対策の難しさが言われて久しい。一方、近年はWF型にとらわれない新しい理論として、ＸＰ（eXtreme programming）に代表されるいくつかの中軽量級開発方法論が、いわゆる、"アジャイル開発"（参考文献1、2、3、4、8、9、10）として広く一般に知られている。ちなみに、アジャイル開発に対する世間の認知度合いを知る一つの参考として、代表的な方法論の名称をYahoo！（JAPAN）で検索した結果は表１のとおりである。

表１　Yahoo!での検索結果

検索条件	2004年12月13日時点	2006年2月26日時点
アジャイル＆XP	5，013件	約12，000件
アジャイル＆スクラム	212件	約20，000件
アジャイル＆FDD	150件	663件
アジャイル＆クリスタル	92件	388件

2.2.1. アジャイル開発で短工期開発に対応する！？

開発手順や成果物を厳密に定義したRUP（Rational Unified Process）に代表されるプロセスを重視する重量級の開発方法論に対し、アジャイル開発はプロセスの簡素化と素早い実装に重点をおいた比較的小規模の開発チーム

向きの中軽量級開発方法論である。アジャイル開発では、顧客の要求と開発チームの体力のバランスをとり、機能を小分け（インクリメンタル）にし、小刻みな（イテラティブ）本番リリースを繰り返すことで、短工期化に対応する実践方法を提唱している。つまり、一般的にはアジャイル開発はインクリメンタルでイテラティブなプロジェクトでは生産性向上の効果が期待できる。（注記：本論文では参考文献1〜4、10のアジャイル開発方法論を参考にしている。）

2.2.2. アジャイル開発を適用するに当たっての障壁

アジャイル開発が提唱する方法論の内容を要約すると『インクリメンタルでイテラティブな開発による短期リリースの繰り返しで、ユーザからの迅速なフィードバックを実現し、開発チームの体力とユーザの要求を調整して次の開発範囲を決め、迅速・確実にリリースすることでユーザ満足度を向上する』ということである。しかし、実際には全体の工期が６ケ月程度で内部設計以降の開発期間が３ケ月程度に制限されるような案件では（特に、Ｗｅｂ技術を基盤としたシステム開発ではその傾向が顕著であるが）複数回のイテレーションを実施することは非常に難しい。つまり、実際のプロジェクトでは一回のリリースでユーザが満足できる品質の成果物を納品しなければならないことが多い。また、アジャイル開発で前提とされている『ＸＰの生産性に関する仮説１：時間がたちストーリー（業務処理の最小単位）の数が増えても、一つのストーリーを開発し修正するための工数が実際には増えることはない』が必ずしも正しくないという報告もある。具体的には、イテレーション回数が増加すると開発メンバ内の暗黙知による開発に限界が発生し、これを解消するために一つのストーリーに取り込む機能の数を減らすことになり、結果として、一回のイテレーションでリリースする機能が少なくなってしまうという事例が紹介されている。（参考文献9 107ページ）

2.2.3. 短工期化のポイント

どのような開発方法論でも、システムを速く正確に作るために最も重要なポイントは情報伝達の正確性、実装の生産性、検証の実効性の三つであると考える。ＲＵＰでは多くのプロセスやレビュー等々、アジャイル開発では"全員同席"（補足１）や"ソースの共同所有"（補足２）等々のアイデアが、ソフトウェア品質を高めるための具体的な方法として提示されている。

補足１：顧客（要求確定者）と開発メンバ全員が同じ作業場所に常駐し作業する。

補足２：開発メンバ全員でプログラムソースを共同所有する認識を徹底し、だれでもすべてのソースの改変や機能追加を行うことができるルールやツールを整備する。

2.3. 短工期化の対策を考える上でのポイント

2.3.1. どこに着目すべきか

2.2.3.で述べた情報伝達の正確性、実装の生産性、検証の実効性の三つを考えると、実装の生産性と検証の実効性は、いわゆる「力仕事」の要素が大きいので情報伝達の正確性が高いほど作業工数と成果は正比例する。また、いくら実装や検証の品質が高くても、もともとの要求や仕様が正確に反映されていなければ成果は全く得られない。情報伝達の正確性に問題があるプロジェクトは要求定義の内容が要求確定者から開発者へ正確に伝達されないため問題が発生することが多い。その原因は、開発チーム内でキーパーソンに情報が集中しすぎて、開発の進捗が停滞したり成果物の品質に問題が発生したりするためである。システム開発ではチャンピオンやスーパＳＥと呼ばれるキーパーソンはシステム開発メンバとして不可欠であり、そのようなキーパーソンから効率よく正確な情報を開発メンバに伝達し実装の生産性と検証の実効性をあげることがプロジェクト成功の大きな要因になる。そのためにプロジェクトでは情報の共有化のためのさまざまなルールが設定されるが、失敗プロジェクトでは情報の共有化がうまくできていないことが多い。

2.3.2. 情報の共有化の問題点

情報の共有化は情報が発信者から受信者への一方通行になりがちになるという大きな問題がある。発信者はメールや掲示板などで情報を発信したことで全員に情報が行き渡ったと思い（錯覚？し）フォローを怠ることがある。さらに、短工期開発の状況下では情報の発信者と受信者間で

の双方向コミュニケーションに十分な時間を取る事ができないことも多い。

2.3.3. 情報の浸透化の必要性

プロジェクトメンバ間で情報の精度を高めるには、発信者から受信者へ情報が（あたかも慣習や文化が人から人へ浸透するように）伝達される環境(以降、『情報の浸透化』と呼ぶ）を作り出す必要がある。　情報の浸透化によって、情報（の洪水）の中から必要な情報が必要としている人に確実に伝わり、情報伝達の正確性、実装の生産性、検証の実効性がより一層促進される。

3. 短工期開発に対応するための問題点と方策

システム開発での情報の発生源は発注元のお客様経営者から開発チームの新規参入メンバまでさまざまであるが、以降に開発チーム内に対象を絞り情報の浸透化のための方策を提案する。

3.1. ソフトウェア開発現場での短工期対応の現状

ソフトウェア開発現場ではシステムを『速く正確に作る』ためのさまざまな実装技術上の改善が行われている。また、ISO9000やCMMIを適用し品質向上のプロセスをもつ会社や部署も多い。しかし、プロセスで定めたリスクヘッジの仕組みが十分に機能せずプロジェクトに問題が発生し追加要員を大量投入して問題を解消した事例も多い。一方、問題を解消するために要員を大量投入したが、かえって生産性低下と品質低下が拡大し一層の混乱を引き起こすという悪循環に陥ったプロジェクトも少なくない。

3.2. 情報伝達の正確性向上に重点をおいたソフトウェア開発方法論の提案

2.2.3.節で述べたプロジェクト成功の三つのポイント(情報伝達の正確性、実装の生産性、検証の実効性）の中で情報伝達の正確性が最も重要である。情報伝達の正確性が高くなればあとの二つは必然的に向上する。本節では情報伝達の正確性を高めるための方法に着目した新しい開発方法論を説明する。

3.2.1. 原型はVモデル型のソフトウェア開発

Vモデル型はWF上での上流工程と下流工程の相関を意識したソフトウェア品質検証の考え方として普及しており、上流工程で作り込んだ要求定義や設計の品質が適正なテスト工程で検証されているかを証明する方法として活用されている。参考文献5ではユーザ要求定義、機能要求定義、プログラム設計、実装作業の各工程に対応する受け入れテスト、システムテスト、結合テスト、単体テストの計画を当該上流工程と同時に着手することを提唱している。（参照：図１）　以降に、この提唱を具体的にプロジェクト計画に活用し、総合的なプロジェクトコスト抑制と短工期化の対応のための提案を説明する。

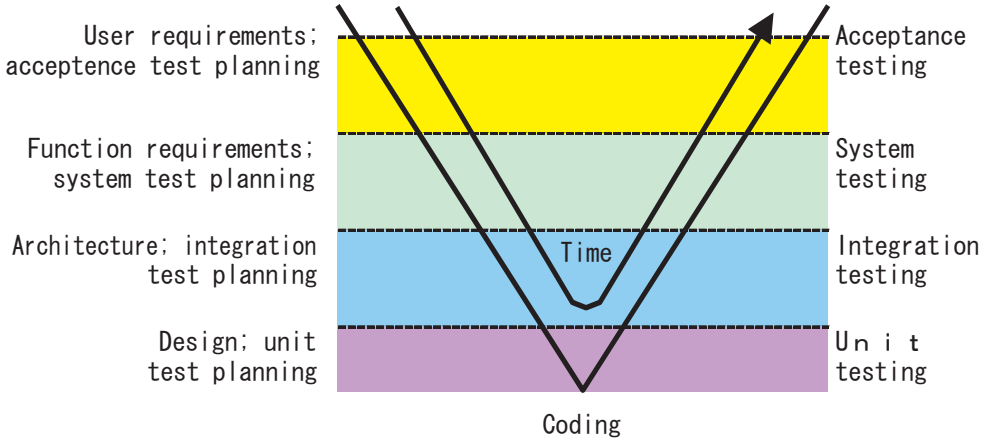

図１　Vモデル開発　テスト計画と設計

(出典：参考文献５ Figure15-1 The V model of software development incorparates early test planning and design)

3.2.2. Ｙモデル型開発－Ｖモデルを応用して上下流工程双方向の同時作業で品質を向上

ＷＦ型の開発では、原則は各作業工程で課題や不明点がなくなるまで作業を継続するが、実際には積み残し課題や不明点は方針レベルまでを当該工程内で検討し一覧表にして次の工程へ進む。このとき、ステークホルダ間で合意した内容の認識にズレが生じ、バグの原因が発生することが多い。方針書や設計書の記載内容のあいまいさに起因して『お客様から見れば不具合、実装チームから見れば仕様変更』という事態が生じる。Ｖモデル型の問題の一つはＶ字上で相関する上流工程と下流工程の時間的な開き（図１のTime軸上の間隔）が大きく、要求や外部仕様に不具合があったときには、その発見が非常に遅れてしまうことである。

さらに、上流工程と下流工程の時間的な開きが大きくなるほど資料の記載内容のあいまい性が増幅される可能性が高くなる。（あとで読み返してみても意味不明だったり、実情にあわなくなった方針だったりということを、実際に経験された方も少なくないと思う。）

この問題を解消するには上流工程と下流工程の作業時期を時間的に、できるだけ近づける対策が必要になる。上流工程と下流工程のすべての作業を同時進行することはできないが、類似システムの開発経験があれば、上流工程の要求定義→外部設計→内部設計（以降、『機能設計』と記述）と下流工程の受け入れテスト計画→システムテスト計画→結合テスト計画（以降、『テスト計画』と記述）を同時に行うことが可能である。機能設計とテスト計画を同時に行うことで双方の問題点がより明確になり早期に問題を解消できる。

そのうえ、機能設計とテスト計画の作業者を分けることで、テスト計画者が（機能設計者とは別人の目で）要求や仕様の観点から機能設計を常時レビューすることになり、その結果、要求仕様の実現性の品質が向上し手戻りの発生が抑止され全体の生産性が向上する。一般的なＷＦ型では、プロジェクト立ち上げ当初からの既存メンバがテスト計画を担当（a）するか、プロジェクト途中から参画する新規メンバがテスト計画を担当（b）し、外部設計の終了後にテスト計画に着手することが多い。（a）では既存メンバがプロジェクトのキーパーソンであった場合、そのメンバに情報や作業が集中し作業負荷が高くなってプロジェクトバに上流工程検討時の背景や要求の内容が正確に伝達できずにプロジェクト進捗に影響がでることがある。

本論文で提案する方法は上流工程で機能設計を担当する作業者とテスト計画を担当する作業者を同時期にプロジェクトに投入することで、（a）や（b）の不具合の発生を未然に防止し、情報伝達の正確性を向上し、手戻りの発生を事前回避することで、総合的にプロジェクトのコスト抑制と短工期化への対応を実現しようとするものである。トンネル工事は双方向から掘削をはじめ、常時お互いの掘削方向を確認し掘り進め、特定の地点で合流することにより工期を短縮する。同様にソフトウェア開発も機能設計とテスト計画の双方がお互いを意識して同時並行的に作業を進めることで問題点の早期発見と解決が可能になり、手戻りの発生を抑止することで短工期計画の実現性を向上することができる。

ただし、厳しい短工期計画の実現に向けて、機能設計とテスト計画の合流点においては、双方の問題点を明らかにし、それらを解決後、実装に着手すべきである。以降、本論文ではこの考え方をＹモデル型と呼ぶ。（参照：図２）

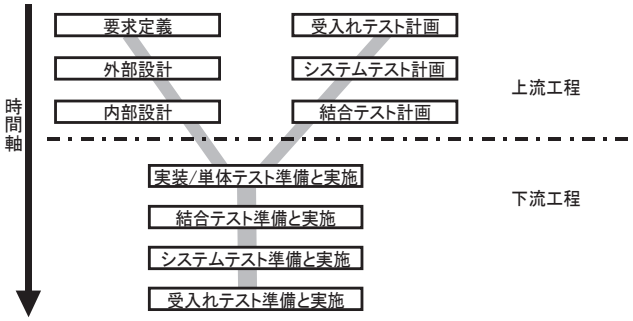

図２　Ｙモデル型ソフトウェア開発

4. Ｙモデルとピアレビューを組み合わせたソフトウェア開発

本章ではＹモデルとピアレビューを組み合わせた開発の具体的な手法や実践に関するアイデアを説明する。

4.1. ピアレビューの応用

ソフトウェア開発での情報伝達の正確性を向上するための有効な手段として開発工程のさまざまな時機でのレビューがある。本論文ではレビューの定義を明確にするために参

表２　レビュー種類

レビュー名称	レビューの概要
インスペクション	顧客(要求提示者)と開発チームでレビューを実施する 成果物作成者以外の開発者が資料を準備し、レビューを受ける 成果物作成者はレビューする側の役割を担う
チームレビュー	開発チームだけでインスペクションを行う
ウォークスルー	成果物作成者がレビューを主導する
ペアプログラミング	二人で作りながらレビューする
ピアデスクチェック	作成者が他の開発者に机上チェックを依頼する
パスアラウンド	所謂、回覧方式によるレビュー依頼
アドホックレビュー	適宜、ステークホルダや有識者が集まって議論する

表３　各種のピアレビューが通常実施するアクティビティ (出典：参考文献６　表3-1)

ピアレビューの種類	計画	準備	アクティビティミーテング	修正	検証
インスペクション	○	○	○	○	○
チームレビュー	○	○	○	○	×
ウォークスルー	○	×	○	○	×
ペアプログラミング	○	×	継続的	○	○
ピアデスクチェック、パスアラウンド	×	○	可能	○	×
アドホックレビュー	×	×	○	○	×

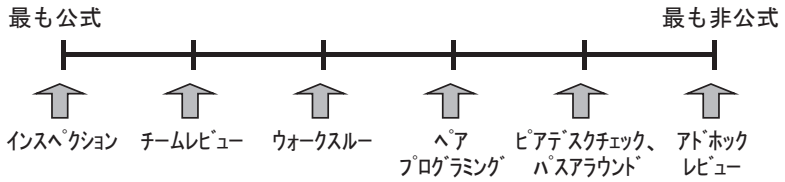

図３　ピアレビューの公式度の分布範囲
（出典：参考文献６ 図3-1）

考文献６に基づいてレビューを６種類に分類して考えている。（参照：表２、表３、図３）　以降、7種類のレビューを参考文献６に倣ってピアレビューと呼ぶ。

4.1.1. ピアレビューを活用した情報の引き渡し

情報をＡの作業者からＢの作業者へ引き渡したときに、情報の漏れ抜けやＢ側の思い込みにより間違いが発生する。これを防止するにはＡがＢの成果物をレビューすることが最善であるが、実際のプロジェクトでは工数やスケジュールの関係で下流工程の成果物を上流工程の作業者がすべてレビューすることは難しい。ピアレビューで提唱している『成果物を作成した開発者の仲間(peer)がレビュイー（レビューされる側）を引き受け、成果物を作成した開発者がレビュアー（レビューする側）になる』という考え方を活用すれば、情報の引き渡しとレビューを同時に実施することが可能になる。

4.1.2. 垂直ペアワーキング

3.2.2.項で"トンネル工事"を比喩にして、Yモデル型では機能設計とテスト計画の作業を分担し同期を取りながら作業を進めることを説明した。一般的なWF型では機能設計とテスト計画の作業は時間軸上で（水平方向に）開きがある。一方、Yモデル型は機能設計とテスト計画の作業を同時進行する。（以降、この作業形態を『垂直ペアワーキング』と呼ぶ。）　垂直ペアワーキングでは相関する機能設計とテスト計画の作業を別の開発者にアサインし、原則は同時並行で作業を進めるが、作業状況に応じて全員で一方の作業に集中する等、動的に作業担当の分散と集中を管理する。こうすることで作業の進捗状況にあわせて動的に工数を再配置しやすくなる。（参照：図４）　この考え方は、クリティカルチェーン（参考文献11）で提唱している"個々の作業予備コストを集め管理することで、プロジェクト全体での最適コストを実現する方法"と類似のものである。

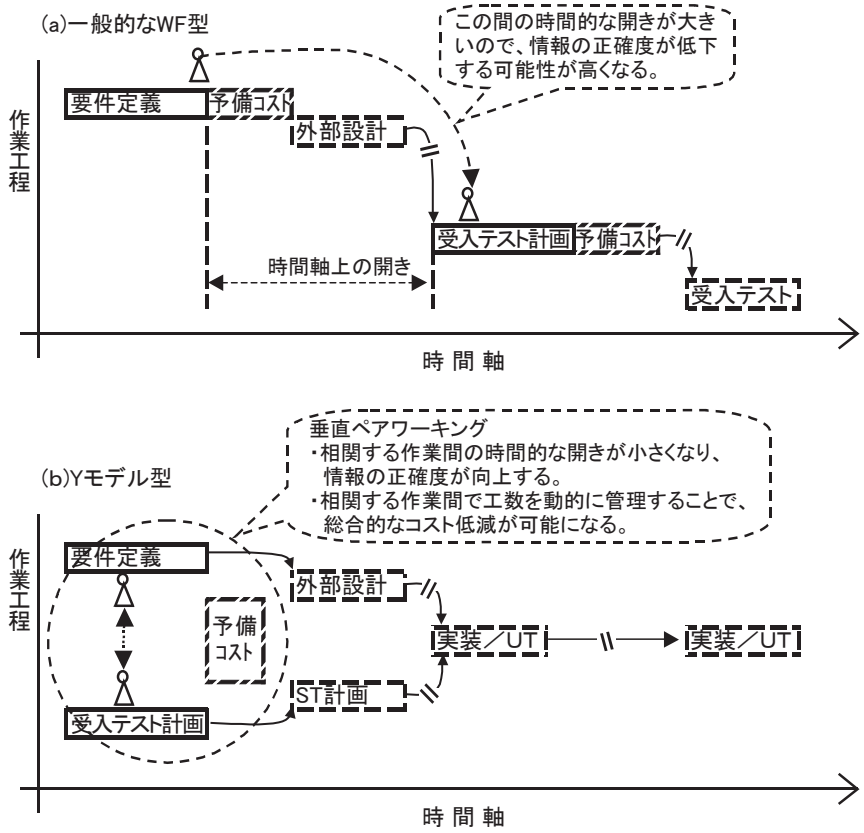

図４　垂直ペアワーキングの考え方

さらに、垂直ペアワーキングでは常にペア相手の作業状況と同期をとって自分の作業を進めるので、ペア間で作業中に自然とアドホックレビューの回数が増えペア間の情報の浸透化が推進される効果が期待できる。

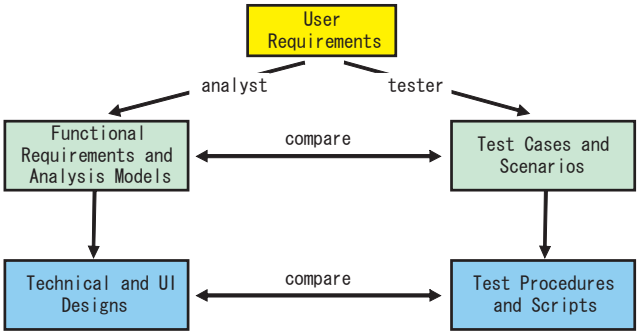

図５　Vモデル開発　開発とテストの作業時期の関係
（出典：参考文献５ Figure15-6 Develpoment and testing work products are derived form a common source.）

4.2. 実践のためのアイデア

4.2.1. ピアレビューによる情報浸透化の促進

ピアレビューではレビューの実効性を確実にするために、インスペクションとチームレビューではレビュー対象物の作成者以外の人がレビュイーとなり、レビュー対象物の作成者はレビュアーを担当する。Vモデル型での要求実現度合いの検証関係（図５のanalystとtesterのcompareの関係）とピアレビューの両方の考え方に沿ってレビューの役割分担を考えると、機能設計の成果物に対するインスペクションやレビューのレビュイーは、テスト計画の作業者が担当することになる。

4.2.2. 垂直ペアワーキングによる工数の抑制

（１）チームレビュー・インスペクションの生産性の向上

筆者の経験上、時間ばかりが長く実効性が低いインスペクションやチームレビューに参加したことが何度もある。このような会議ではレビュイーの説明力不足が原因でレビュアーが資料の内容を理解でき

ないまま説明が終了し、有益な指摘点がほとんどないか、逆に、質問の焦点が十分に絞りきれないまま持ち戻りの宿題が山積されることが多い。垂直ペアワーキングではテスト計画者がレビュイーになって機能設計の成果物のレビューを受けるので、機能設計者からテスト設計者に情報が正確に引き継がれたかの検証の精度が向上し、Vモデル型と比べて問題の発見が早くなる。また、レビューにはペアワーカー以外のステークホルダも参加しているので、ペア間で考え方に対立が発生した場合の調整が迅速にできる。

（2）開発要員数の山積みの平準化による要員コストの抑制

プロジェクトの要員投入時の教育効率は、開発の後半になるほど伝達しなければならない情報が増えるので効率が落ちる。他方、要員調達の面ではプロジェクトへの参加期間が長いほど技術力の高い要員の確保が容易になり、優秀な要員の長期確保は生産性の向上に直結する。図６で示すように、通常のＷＦ型に比べＹモデル型では開発要員の投入が早期に行われその後の山積みが平準化されるので、前述の理由で生産性が向上し要員コストを抑制できる。また、Ｙモデル型では上流工程の作業者から下流工程の作業者への情報伝達の正確性が向上するので、早期に上流工程の開発者をプロジェクトから撤収することが可能になる。また、テスト計画作業の要員をプロジェクト途中で投入する場合、図６－(a) 内の計画要員導入教育の工数が発生するが、図６－(b) ではその工数は不要になる。さらに、開発要員の急激な増加は開発場所（物理的なスペース）の環境悪化に直結しているので、開発要員数の山積みの平準化は作業環境の悪化を防止できるという副次的な効果もある。

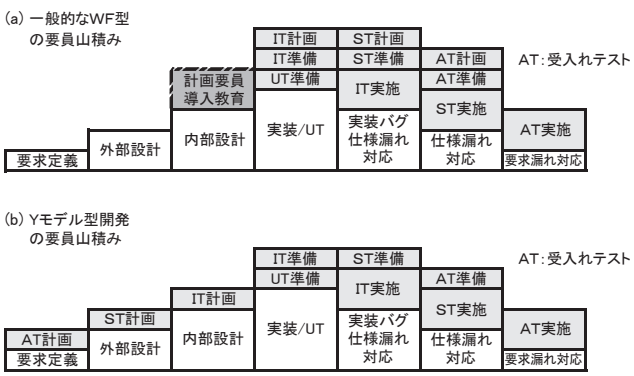

図６　ウォータフォール型とＹモデル型の開発要員山積み

4.2.3. 追跡可能性（Traceability）の追求

情報が発信者から受信者へ正確に伝達できているかを検証するための手段としてソフトウェア品質の追跡可能性が最も重要になる。しかし、『追跡可能性を厳密に実施すると、通常は利点を上回るコストがかかる。』（参考文献8 307ページ）　実際のプロジェクトでは文字どおりには追跡可能性を徹底的に追求することは難しいため、現実解に落とす工夫が必要である。

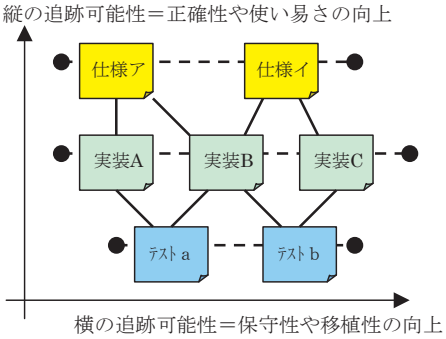

図７　縦の追跡可能性と横の追跡可能性

本論文では追跡可能性を、縦の追跡可能性（ニーズ⇔要求⇔仕様⇔実装⇔テスト間での仕様の連続性：図７の実線）と横の追跡可能性（同一工程の成果物間の関連性や影響度：図７の破線）に分類して考える。縦の追跡可能性は要求実現の正確性や使いやすさに関連し、横の追跡可能性は保守性や移植性への関連が大きい。要求定義に関する情報伝達の正確性を向上するには縦の追跡可能性が重要である。

Ｖモデル型では上流工程と下流工程の時間の開きが大きいので、縦の追跡可能性が途切れる状態（例：資料の行間の情報落ち等）や下流から上流に遡ることができない状態（例：実装に気を取られ、元の仕様がうまく反映できていないプログラムを作成してしまう等）が発生する可能性が高くなる。その結果、『テストa実施時に仕様自体に不都合が発見されたが、仕様の背景や仕様と実装の相関関係が不明確なのでテスト結果が正しいかどうか判断できない。』という不具合が発生することがある。Yモデル型では機能設計とテスト計画を同時進行で作業するので、このような不具合の発生率を抑止することが可能である。

4.2.4. ユースケースをテストケースに活用

分析や設計フェーズでユースケースを作成するプロジェ

クトも多い。一般的には『ユースケースは、システムにとってあらかじめ用意された機能テスト記述となる。』（参考文献7 222ページ）　しかし、Vモデル型では要求定義・機能設計工程とテスト計画・テストシナリオ作成工程との時間の開きが大きいため、ユースケースの紙面上に書き切れなかった（いわゆる、行間に書かれた）情報が時間の経過に比例して希薄になる。また、実装工程後または実装工程と並行で機能テストの内容を記述すると、現場の開発者レベルでは実際に実装したプログラムの処理（ふるまい）の検証に目が注がれがちになる一方で、当該機能の発生源である要求の検証を怠りがちになる傾向が見られる。その結果、開発チーム内のテストは合格でもお客様テストで障害となって戻ってくることがある。Yモデル型では、機能設計とテスト計画の作業時間の開きによって発生する情報伝達の不正確さを小さくすることができるのでユースケース上に記述された要求や仕様の不明点の解消がピアデスクチェックなどで容易になる。また、ユースケース作成時に機能設計者が作成した汎用シナリオ７）（アクタ名やデータ値の代わりに「顧客」や「住所」といった変数名を使って記述したシナリオ）をテスト設計者が具象シナリオ７）に書き直すことでウォークスルーレビューを実施したのと同じような効果がえられる。

4.3. スケジュール作成上の留意点

図６で示したように、Yモデル型はテスト計画を前倒しするのでVモデル型よりも全体スケジュールを短縮できる可能性がある。反面、Yモデル型は機能設計が終了する度に、テスト計画への影響を見直す必要があるのでVモデル型よりレビューの時間が多くなる。また、Yモデル型はVモデル型と比べると開発要員数の山積みが平準化され、かつ早期にチームメンバ全員の作業が開始されるので分析者、設計者、実装者、テスト者等、開発の全工程の要員手配を早めに行う必要がある。

4.4. チーム立ち上げ時の留意点

4.4.1. テスト実施工程のリーダーをプロジェクトの中で育成

Vモデル型の開発スケジュールでは上流工程の作業者が下流工程の作業チームのリーダーになることが多い。しかし、この方法は実質的なリーダーが不在になるリスクを持っている。具体的な例として『上流工程作業のキーマンがテストチームリーダーとなって計画を立て、テスト実施は新規の開発メンバに任せる体制をとったが、テスト期間中に上流工程のやり残しやテストで発見された漏れ抜けの対応でテストリーダーが上流工程の手直し作業に忙殺され、テスト進捗のボトルネックとなってしまった』というような事態が発生することがある。このような問題を解消するには上流工程のキーマンをテスト実施期間中はフリーの立場にしておくことが最善であるが、実際のプロジェクトでは要員コストの問題もあり、リーダーを任せられる要員を希望通りに手配することは難しい。テスト実施リーダーをプロジェクトの中で育成する場合、Yモデル型はテスト計画作業が早期に開始されるのでＯＪＴの期間を長くとることができ、プロジェクト開始当初にリーダーに予定された要員にＯＪＴの目的を指示することで育成が容易になる。

4.4.2. 実装者とテスタでソース共同所有

開発チーム内の結合テストの場面では、テスタがバグを実装者の前で再現して見せることが多い。両者がそのままデバッグに入ることもしばしば見られる。両者が１台の端末の前で作業している様子は一見、ペアプログラミングのようであり、テスタがプログラムの内容を十分に理解し、また、両者にソース共同所有の責任（自覚）があれば、生産的な意見交換が行われ作業時間に無駄がなくなり二人分の生産性が期待できる。しかし、実際は、二人の作業者のどちらか一方だけが考え、作業する時間が長く、結果的には一方の作業者が作業待ちで時間を浪費することが多い。テスト中に発見したプログラムのバグをテスタが実装者に修正依頼することなくテスタがその場で修正できれば、この浪費を防止できる。具体的な実践としては開発者とテスタでペアプログラミングを行うことが最善だが、ペアプログラミングの実践は環境整備や立ち上がりの要員コストがかかるので実際には難しい。簡便的な方法としてソースの共同所有（2.2.3.節の補足2）を実践することだけでも作業時間浪費の防止効果が期待できる。具体的な例として、複数の開発者で一つのプログラムを修正するという認識をチーム全員に徹底すれば、ほかの開発者が読みやすいコードを書くようになる。また、コーディング規約が守られているかのチェックが、常時、ピアデスクチェックされることになり、その結果、チーム全体でコーディング品質向上

の意識が高まる。

5. おわりに

経験豊富なＳＥは上流工程の作業中に下流工程の課題を常に意識している。上流工程の作業中に『これは、どうやってテストするのだ？』とか『どこまでテストするつもりか？』などを考え、想定される課題を解消しておくことでプロジェクト全体での生産性や品質を上げるやり方は開発経験に裏打ちされた貴重な知恵であり、この知恵がＹモデル型ソフトウェア開発のアイデアの種である。また、本論文の読者の方はレビューの重要性を十分に理解し、日々、実践されているであろうと思われる。本論文で提案したＹモデル型ソフトウェア開発とピアレビューの応用を参考にしていただき、一つでも多くのプロジェクトが成功することを切望している。

参考文献

1) ケント・ベック
"XPエクストリーム・プログラミング入門―ソフトウェア開発の究極の手法"
ピアソンエデュケーション(2000/12)

2) スティーブンR・パルマ＋ジョン・M・フェルシング
"アジャイル開発手法FDD―ユーザ機能駆動によるアジャイル開発"
ピアソンエデュケーション(2003/03)

3) ケン・シュエンバー＋マイク・ビードル
"アジャイルソフトウェア開発スクラム"
ピアソンエデュケーション(2003/03)

4) アリスター・コーバーン
"アジャイルソフトウェア開発"
ピアソンエデュケーション(2002/08)

5) Karl E. Wiegers
"Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle (Pro-Best Practices)"
Microsoft Pr 2nd 版 (2003/02/26)

6) カールE．ウィーガーズ
"ピアーレビュー　ー高品質ソフトウェア開発のためにー"
日経BPソフトプレス(2004/03/01)

7) アリスター・コーバーン
"ユースケース実践ガイド　効果的なユースケースの書き方"
株式会社翔泳社(2002/12/05)

8) スコット・W・アンブラー
"アジャイルモデリング"
株式会社翔泳社(2003/08/08)

9) バリー・ベーム＋リチャード・ターナー
"アジャイルと規律"
日経BP社(2004/8/9)

10) ケント・ベック
"テスト駆動開発入門"
ピアソンエデュケーション(2003/09/15)

11) エリヤフ・ゴールドラット□
"クリティカルチェーン"
ダイヤモンド社(2003/10/30)
