Software Factory適用プロジェクトの課題
－ProjectTower®検証事例報告－

テクニカルコンピテンシー部
シニアITアーキテクト

堀　扶

Tasuku　Hori
tasuku-hori@exa-corp.co.jp

近年、オフショア開発の生産性を向上させる手段としてSoftware Factoryの適用を検討する企業が現れ始めている。当社でも本戦術の有効性を検証するために、自社開発ツールであるProjectTowerを用いてSoftware Factory環境を構築し、開発標準に完全準拠したプロジェクト運用を実際に行った。本論文では、検証結果をもとにSoftware Factory適用時の課題について考察する。

1. はじめに

近年、日本においてもオフショア活用を前提とするプロジェクトが増加しつつある。しかし、距離や文化・言語の壁が存在するため、生産性が低下しコストメリットを享受できないことが問題となっている。そこで、国内SI'er数社は、オフショア企業との間にSoftware Factoryを構築し、生産性の低下を可能な限り防ぐことで、コストメリットを維持しようと検討をはじめている[1) 2)]。

Software Factoryとは、プログラミング品質および生産性を向上させるために必要な標準的なツール、成果物、およびマネジメントデータを管理するためのデータベースを共有する、ソフトウェア開発・販売・工程管理・原価管理・調達管理・品質管理を統合した組織のことを指す[3)]。

当社においてもこのような戦術の有効性、実現可能性を検討するために、Software Factory環境を構築し、実プロジェクトへの適用を行った。本稿では、その検証・評価結果について報告する。

2. ProjectTower

当社では、Software Factoryに注目し、この環境を提供する開発ツールProjectTowerの開発を進めてきた。

ProjectTowerは、2003年より検討、設計、実装してきたプロジェクト支援のためのツールである。

ProjectTowerには、３つのモデルが登録されている[4)]。

・基準計画（プロジェクトライフサイクルモデル）
・仕様間の関連（トレーサビリティモデル）
・成果物テンプレート（成果物モデル）

ProjectTowerは、これらのモデルに従い、以下の３種類のワークフローをプロジェクトメンバに提供する。

・成果物の作成
・成果物の変更

プロジェクト内にて作成する成果物やレビュー報告書、変更依頼書は、ProjectTowerからテンプレートとして提供される。これらテンプレートを使用していない成果物を提出した場合、ProjectTowerは受領を拒否する。また、仕様間の関連を考慮していない成果物を提出した場合も同様である。

これらの機能により、ProjectTowerを使用することで、開発標準に対する統制が可能となる。

またProjectTowerはソフトウェア開発における工程管理・原価管理の一部の機能を提供する。これらを元に、プロジェクト状況の様々な検証ができる。

3. 検証対象プロジェクト

Software Factory環境がプロジェクトに与える影響を明らかにするため、社内システムの開発プロジェクトを対象に検証した。対象プロジェクトの概要を表１に示す。

表１　検証対象プロジェクト

プロジェクト名	社員保有技術管理システム
概要	SEの能力開発やリソース管理を目的にした、社員の保有スキルを管理するシステム
開発期間	2008年2月～6月
フェーズ	要件定義～システムテスト
予定工数	25人月
参加予定人数	5人
画面数/帳票数	17画面/2帳票
アクタ数/	5アクタ
ユースケース数	25ユースケース
アーキテクチャ	2層Webアプリケーション TransactionScript+DAO Struts/Spring/iBATIS/Acegi ProjectTowerを使用

選定の理由として、プロジェクト規模が比較的小さく、かつプロジェクト実績が比較的多いアーキテクチャを採用していることがあげられる。

4. 検証結果

Software Factory導入の目的は、品質および生産性の維持・向上にある。よって、検証対象プロジェクトの運用結果を、品質および生産性の２つの視点で集約することとした。

なお、2008年8月31日現在、検証対象プロジェクトは完了していない。このため、現時点の状態でデータ集計している（詳細設計で作成する成果物の42%まで完成）。

4.1. 品質

実行中のプロジェクトに対して、品質を定量的に評価するために、以下の指標を用いる。

・欠陥発見数、除去数
・品質分析傾向

4.1.1. 欠陥発見数、除去数

レビュー実施時の欠陥発見数と除去数の累積状況を図１に示す。

欠陥発見数と欠陥除去数がほぼ同じ値であるため、グラフが重なっている。レビューにて発見した欠陥は、ほぼ同時に対応作業を行っていることがわかる。

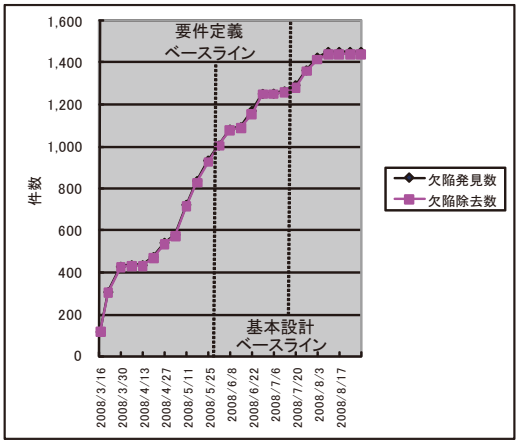

図１　欠陥発見数・除去数

4.1.2. 品質分析傾向

検証対象プロジェクトでは､各フェーズ終了時にQI(Quality Inspection:品質検査) [5] を実施した。図２は、要件定義時のQIコメントシートの品質傾向分析より抜粋したレーダチャートに、当社QI上流工程平均値を重ねたグラフである。

これらの指標は５点評価となっており、網羅性については満点の評価である。整合性、詳細性、安定性についても当社平均より上位の評価であった。

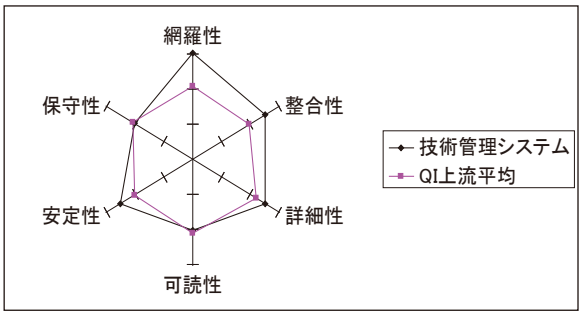

図２　QI結果(要件定義)

なお、基本設計や詳細設計に対するQIは執筆時点では完了しなかった。

4.1.3. 品質まとめ

欠陥をほぼ同時に解決していること、要件定義のみであるがQI結果が当社平均以上であることから、弊社の現在の品質検査基準に対しては、検証対象プロジェクトでは一定の品質は維持可能であることが確認できた。

4.2. 生産性

生産性を検証するために、表２に示す４つの指標を使用した。

表２　検証に使用した指標群

指標	概要	単位
SV	スケジュール差異(Schedule Variance)。基準計画に対するプロジェクトのスケジュール遅延を表す。SV=-10の場合、10人日分の作業が遅延している状態である。	人日
CV	コスト差異(Cost Variance)。基準計画に対するコストベースの遅延を表す。CV=-10の場合、10人日分のコストが超過している状態である。	人日
CPI	コスト効率指標(Cost Performance Index)。基準計画に対する評価時点のプロジェクトメンバの平均生産性比率。CV=80の場合、基準計画での見込み生産性の80%しか発揮していない状態である。	%
TCPI	完了までのコスト指標(To Complete Performance Index)。 CPIと異なり今後求められるコスト効率を表す。TCPI=130の場合、遅延を回復するために今後1.3倍の生産性を求められている状態である。	%

SVとCVは、生産性評価のための指標ではないが、今回はプロジェクト進捗推移把握のために使用した。

CPIとTCPIは、参考文献[6]においてProject Control Panelの生産性を評価するための指標のベストプラクティスとして紹介されている。

4.2.1. 基準計画との差異

図３は、SVとCV推移を表したグラフである。

SVの推移から、要件定義後半からともに大きく差が開

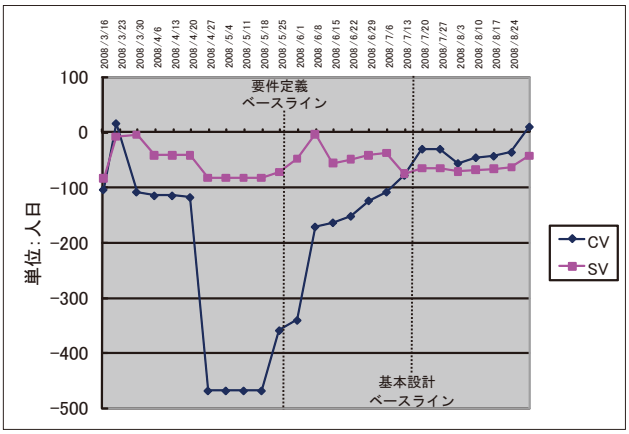

図３　SVとCV

いているが、基本設計初期に進捗が回復した。しかし再度遅延が発生したことがわかる。特に進捗が遅延していた時期が要件定義後半から基本設計初期である。ただし要件定義ベースライン通過時に進捗の一部が回復している。

要件定義後半から基本設計初期のCV推移は、SVとほぼ比例しており、コスト遅延が回復していないことがわかる。これは、この時期の進捗がしばらくなかったことを表している。

基本設計初期のスケジュール回復時期のCVは-172人日であり、コストとしてはまだ回復していない。基本設計中盤以降は、徐々にCVを回復させており、遅延に対するキャッチアップ作業が少しずつ進んでいることがわかる。

4.2.2. コスト効率

各フェーズにおけるCPIおよびTCPIを計測した結果を図４に示す。

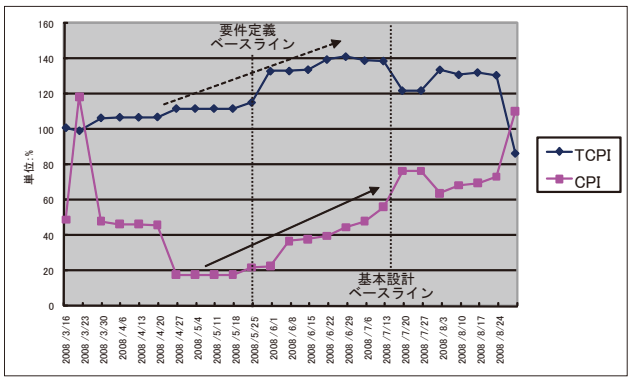

図４　CPIとTCPI

要件定義後半から基本設計初期にかけて、CPIがプロジェクト全体の下限値のまま推移した時期が約１ヶ月続いていたことがわかる。基本設計中盤より生産性が向上し、要件定義終了時点の22%から56%まで向上させている。

要件定義後半から基本設計初期までの期間、プロジェクト全体でCPIが最も低かった（図４実線矢印）。しかし、TCPIは基本設計中盤以降増加している（図４破線矢印）。これは、生産性を向上させても、求められる生産性がより厳しくなっていることを示している。よって、生産性以外の問題についても考察する必要がある。

4.2.3. 生産性まとめ

要件定義から基本設計初期までの生産性が低く、特に要件定義後半の約１ヶ月は進捗が進んでいない。基本設計中盤から徐々に生産性が向上するが、基本設計初期までの生産性を挽回するまでにかなりの時間を要していた。

また、原因として生産性以外の問題が潜む傍証も発見できた。

4.3. 検証結果まとめ

品質および生産性に対して検証を行った結果、検証対象プロジェクトでは以下の２点の効果を確認した

・品質劣化低減効果は得られた

・生産性は低下した

特に、生産性については、以下の状況を確認することができた。

・要件定義後半１ヶ月の進捗がほぼ停止

・要件定義から基本設計初期の生産性が低い

5. 考察

検証結果をもとに、その原因について以下に考察する。

5.1. 品質維持効果

ProjectTowerでは、すべての成果物に対してテンプレート（図５）の利用を強制する。テンプレートにより、章節単位での記述の強制だけでなく、成果物間のトレーサビリティについての記述も強制する。このため、各仕様記述は、必要かつ十分な内容で記述をしなくてはならない。品質維持効果が得られたのは、このためである。

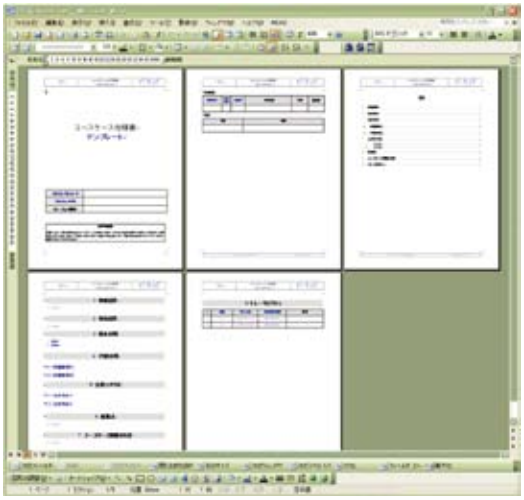

図５　成果物テンプレート一覧(Word)

しかし今回はProjectTowerが提供する成果物テンプレートの検証も同時に行った。そのため、章節編成が実用に合致しないなど、可読性や保守性についての評価は、平均的な品質レベルにとどまる結果となった。

Software Factory環境であっても、成果物テンプレートを含む開発標準の充実、レビュー技術の研鑽など品質向上のための努力は、継続的に実施していく必要がある。

5.2. 生産性低下

プロジェクトメンバの生産性以外の問題が内在すると推測されるため、さらに調査し、原因を特定した。

5.2.1. 要件定義後半の進捗停止

表３は、要件定義での成果物作成開始、成果物完成、そしてベースラインチェックまでの予定および実績日付と遅延日数をまとめたものである。

表３　要件定義における遅延状況

	予定日	実績日	遅延日数
成果物作成開始	2/18	2/18	0営業日
成果物作成完了	3/14	3/26	7営業日
ベースライン チェック完了	3/17	5/28	63営業日

ベースラインチェックとは、ProjectTowerの機能の一つで、成果物へ記述された各仕様の関連が正しいかを確認する自動チェック処理のことである。ベースラインチェックでは、ProjectTowerに登録されたモデルをもとに成果物をすべてチェックし、不整合がないことを確認する。ベースラインチェックが不合格のままでは、次のフェーズへ進むことができない。

要件定義では、このチェックを通過するまでに63日間（表３網掛け部の期間）を要した。その際の主な作業は、要件定義全成果物に対する、トレーサビリティ表の修正、ProjectTower自身の障害対応であった。

(1) トレーサビリティ表の修正

ProjectTowerを使用する際、プロジェクトメンバは、成果物作成の際に、トレーサビリティ表と呼ばれる仕様間を表す表を、成果物に添付しなくてはならない(図６)。

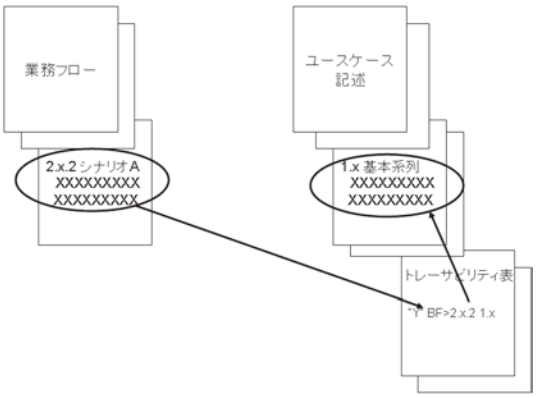

図６　トレーサビリティ表の記述

図７は、ユースケース記述書のトレーサビリティ表である。一次元の表で、下流の成果物であるほどページ数が増大し、プロジェクトメンバの入力ミスが多発した。

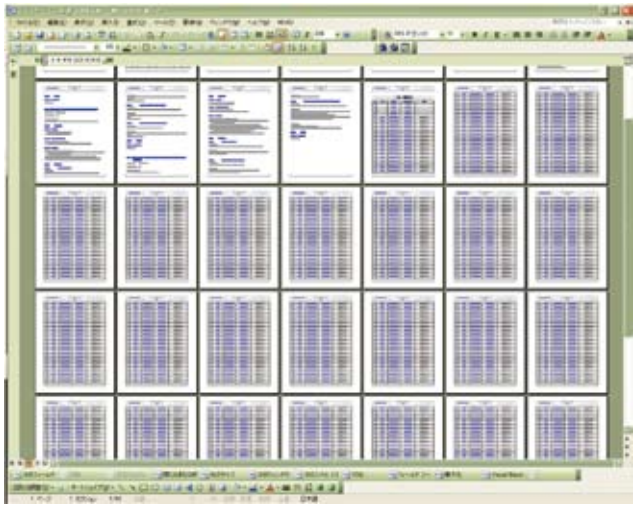

図７　旧トレーサビリティ表

基本設計後半より、この一次元の表形式から、視覚的にチェックを行いやすい二次元の表形式へと改良を行った(図８)。可読性・保守性が向上したため、レビューアによるトレーサビリティチェック作業もかなり低減することができた。

(2) ProjectTowerの品質

ProjectTowerのプロジェクト適用は、本検証作業がはじめてであり、特に要件定義ではProjectTowerの障害対応が遅延の大きな原因の一つであった。実際、ベースラインチェック通過のために充てた63日間のうち、約１ヶ月間はProjectTowerを停止させ、障害対応作業を実施した。

検証対象プロジェクトのSV、CV、CPIが要件定義後半フラットなデータが連続していたのはこのためである。

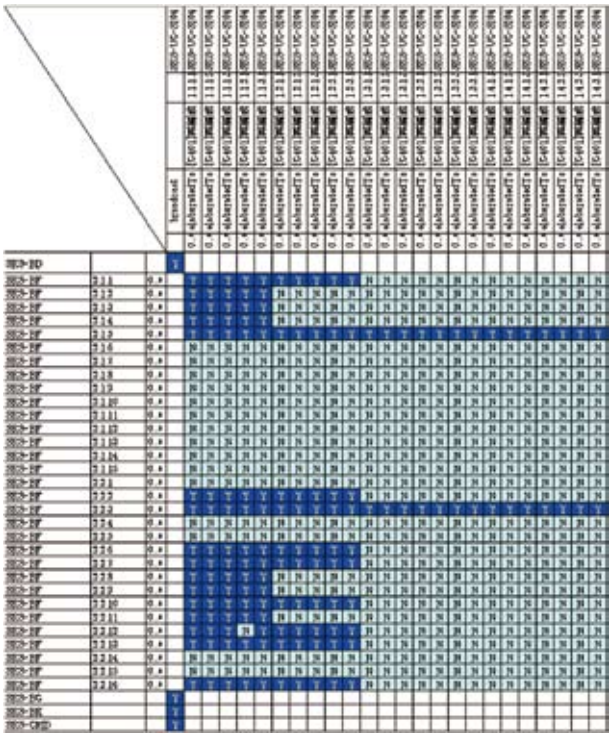

図８　新トレーサビリティ表

なお、現在ではProjectTowerの品質は実用レベルまで向上している。基本設計終了時では、５日間でベースラインを通過した（表４網掛け部の期間）。

表４　基本設計における遅延状況

	予定日	実績日	遅延日数
成果物作成開始	3/17	5/28	48営業日
成果物作成完了	5/16	7/11	41営業日
ベースライン チェック完了	5/19	7/16	42営業日

5.2.2. 要件定義から基本設計初期の生産性低下

特に生産性低下が著しい要件定義から基本設計初期までを考察した。

(1) 割り当て可能時間と総作業時間

図９は、プロジェクトへの割り当て可能な時間と、実際のプロジェクトにて作業に使用した作業時間の累積グラフである。

要件定義ベースライン直前の作業時間の急増は、ベースラインチェックにて不合格となったトレーサビリティ表の修正作業の時間をまとめて登録したためである。

ベースラインチェック対応作業を除くと、要件定義全般

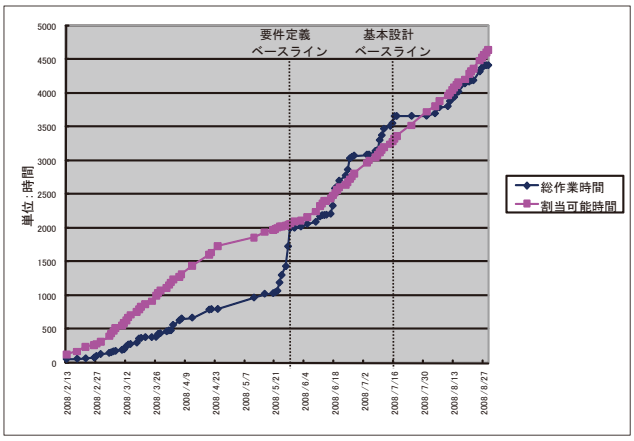

図９　割り当て可能時間と総作業時間（累積）

において、割り当て可能時間と総作業時間の差が大きい。これは、待機している要員が多く存在していたことを示している。

また、基本設計初期の状態でも、待機時間の余裕がみられる。要件定義でのベースライン通過作業がなければ、要員の余剰は継続していた。

基本設計中盤からは、作業時間が増加し、要員ほぼ全員が高い稼働率で作業に従事している。

(2) プロジェクトメンバ別総作業時間

さらに詳しく調査を行うために、プロジェクトメンバ(外部レビューアを含む)あたりの作業時間、レビュー時間をグラフ化した（図10)。作業時間は、成果物作成に費やした時間の合計、レビュー時間は、レビュー準備を含むレビュー実施時間合計である。

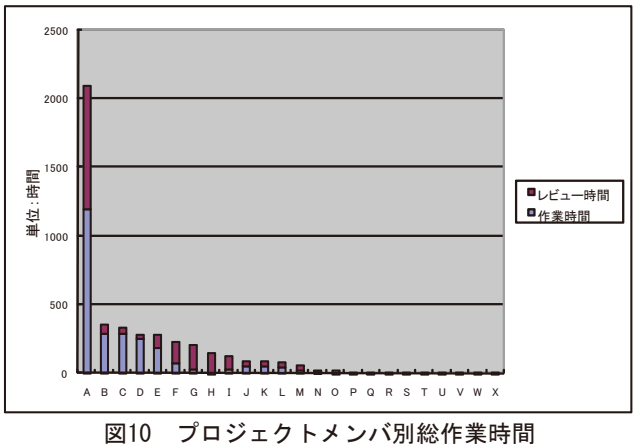

図10　プロジェクトメンバ別総作業時間

この検証対象プロジェクトは一人のメンバーに作業負荷が極端に集中していることが判明した。

(3) プロジェクトの首

要件定義から基本設計前半にかけて、負荷が集中したメンバの担当したタスクは、アーキテクチャ設計である(図11)。アーキテクチャ設計では、機能要件、非機能要件をもとにシステムのアーキテクチャを決定する作業である。機能要件、非機能要件が集中する本作業を「プロジェクトの首」と呼ぶことにする。

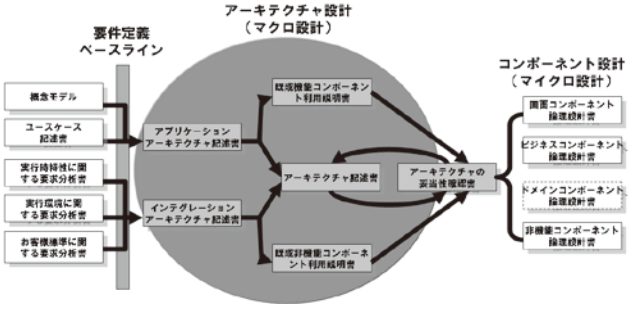

図11　プロジェクトの首

要件定義でも業務フローやユースケース記述などプロジェクトメンバで分担可能な作業は存在するが、アーキテクチャ設計は一定以上のスキルを保有するメンバが必要となる（図12）。

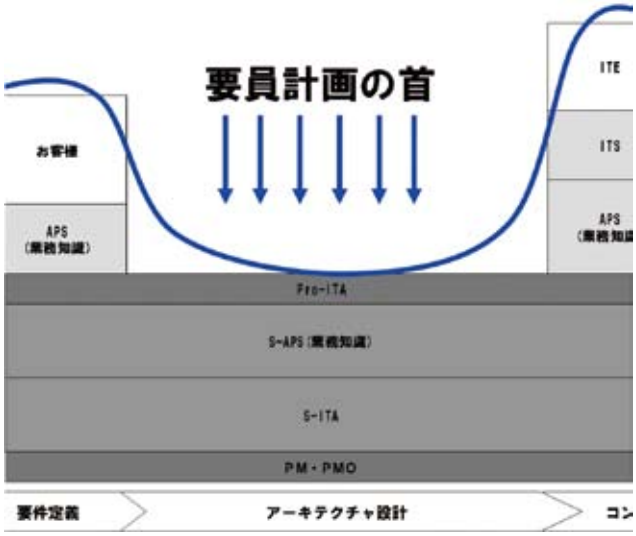

図12　要員計画の首

検証対象プロジェクトでは、主要なアーキテクチャ作業を一人で担当したため、次のタスクが開始できず、結果遅延を広げる結果となった。

スキルを保有しないメンバを割り当てたと仮定した場合、通常はアーキテクチャ設計が不備のまま後続タスクへと進み、問題の顕在化がプロジェクト後半になると予想される。Software Factory環境下の場合は、今回のプロジェクトのように、アーキテクチャ設計時点でのプロジェクト遅延として問題が顕在化する。

いずれにしても要員配置の問題は解消できないが、Software Factory環境下では問題が早期に発見できるという利点がある。

5.3. 考察のまとめ

Software Factory（ProjectTower）環境下における課題と、Software Factory環境下での検証により重要性を再確認したプロジェクト共通の課題の２つに分類する。

5.3.1. Software Factory環境における課題

本稿のグラフや表は、図２を除き、ProjectTowerが保有するデータ（図13）をもとに集計を行った。これらのデータはリアルタイムで計測可能なデータである。常時指標を集計し、常時監視を行うことで、品質向上や生産性向上のための施策をプロジェクト活動中の早い段階で投入することができる。

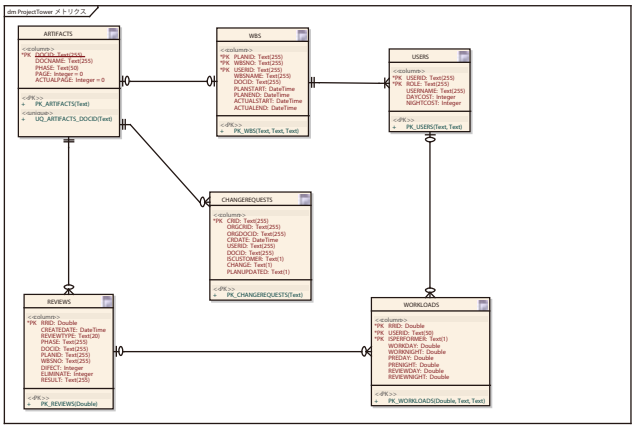

図13　ProjectTower保有データの一部

2005年頃よりソフトウェアメトリクスの重要性が議論されてきた。ソフトウェアメトリクスとは、ソフトウェア開発をさまざまな視点から定量的に評価したものを指す[7)]。しかし近年統合開発環境に搭載されたメトリクス機能は、ソースコードやテストを中心とした指標が多い。

反面プロジェクトに対するメトリクス（ここでは「プロジェクトメトリクス」と呼称する）に対する議論はソフトウェア企業内にとどまっている[8)]。

今後はプロジェクトメトリクスのベストプラクティスについて検討し、ProjectTower上のデータを有効活用できる機能の充実をはかる必要がある。たとえば、いくつかのプロジェクトメトリクスに対してしきい値などのルールを定義し、ルール違反を自動検知する、などである。さらに、検知の際には定められた社内規程にあわせたワークフローを実行する仕組みなどを組み込むと、品質や生産性悪化時の早期発見・早期対応できる。

5.3.2. プロジェクト共通の課題

品質を維持するためには、より強固な開発標準およびレビュー技術が必要である。今回の検証では初適用となる標準を使用して不具合が多かったため、適用実績件数の多い”枯れた”標準類を整備する必要性を痛感した。

また、生産性を維持するためには、プロジェクトの首を担当可能な要員の確保が最も大きな課題である。特に幅広い知識・経験を保有するメンバを育成、確保し続ける必要がある。

特にSoftware Factory環境では、プロジェクトの遅延として顕在化する。プロジェクト早期での対処を要求されるため、優先度はより高くなる。

しかし、残念ながらそのようなスキルを保有するメンバは少なく、現実的にプロジェクト計画時に確保することは難しい。

このため、要員育成という長期的施策だけでなく、不足するスキルを補う短期的施策を並行して講じる必要がある。スキルを補填する手段として、アーキテクチャや基盤などのベストプラクティスの再利用、ビジネスパートナを含むプロジェクトメンバの類似プロジェクトへの連続登用などが考えられる。

6. さいごに

オフショア開発をビジネス戦術として成功させるには、どの作業範囲を依頼するかを検討する必要がある。たとえば、既に述べたプロジェクトの首以降を依頼する場合、品質や生産性に対するリスクは低くなるが、コストメリットを失う。また、要件定義以降を依頼する場合、コストメリットは大きくなるがプロジェクトリスクは増大する。ここで、リスク低減のためにレビュー作業を手厚く行うと、結果コストメリットが低下する。また、要件定義から受注するための営業戦略も必要である。

こういったビジネス戦術の検討には、プロジェクトメトリクスの蓄積と分析が重要である。様々なタイプ(規模、期間、アーキテクチャなど)のプロジェクトのメトリクスを蓄積・分析した上で、当該プロジェクトのための最適な計画を作成し、実施期間中は発生した様々な事象を対策後の効果とともに定量的に記録することが不可欠である。SI'erが、実現可能なビジネス戦術を作成・遂行する上で、最も重要な課題は、プロジェクトメトリクスの蓄積と分析にある。

そして、Software Factoryの構築は、プロジェクトメトリクスを収集するための解決法として、有効な戦術の一つである。

参考文献

1) 渡辺純，ビジネスアプリケーション開発の工業化，第7回ソフトウェアファクトリ研究会資料，2007年
2) 大力修，ソフトウェア開発を近代製造業へ，ソフトウェアファクトリ研究会資料，2008年
3) 松本吉弘，ソフトウェアファクトリ序説, ソフトウェアファクトリ研究会資料，2007年
4) Keiichi Matsuyama, Software Project Driven by the Development Standard Modeled by UML Model-Driven Adaptive Development/MDAD, ProMAC 2008 – Anchorage Alaska, 2008
5) 細川宣啓,"Smoke Sensor System" としての Quality Inspection の適用事例：品質インスペクション技法適用によるソリューション・リスクの最小化と成果物品質の向上例, プロジェクトマネジメント学会研究発表大会予稿録 Vol.2003, No.春季(20030311) pp. 279-284, 2003年
6) Norm Brown, The Program Manager's Guide To Software Acquisition Best Practices, Software Acquisition Best Practices Initiative, 1998
7) 長瀬嘉秀/新船弘之, ソフトウェアの品質を数値化して確かめる, アットマークIT記事, 2005年7月22日
8) 俵宏利, プロジェクト管理におけるメトリクスについて, NSA-Vol.8 No.08007, 2008年3月26日
