アーキテクト視点の体系化

Schematizing Perspectives for Information System Architects

千葉 浩太郎

要 約 情報システムは，現代社会において欠かすことのできない社会基盤である．メインフレームが中心的な存在であった時代と比べてオープン系の情報システムでは自由度の高まりによって複雑性が増した．さらに，近年ではシステム開発技術の進化も急速である．本稿では，システム開発技術の変遷に影響されにくい基礎的，普遍的な部分として，アーキテクトが持つべき視点や観点の体系化について注目する．その上で情報システムの機能的・非機能的な側面を捉えるこの視点体系を，今後どの様に活用できるかについて考察する．

Abstract The information systems are becoming a social infrastructure in modern society. Compared to years in which the information system market was dominated by mainframe computers, the open systems are growing increasingly complex, and the technology for the system development is also changing rapidly. In this paper, we focus on schematizing the system development perspectives which are important for the computer architects in fundamental and universal part of the technology for information system development. Then, we examine how the body of perspective, which encompasses the functional and non-functional aspects of the information systems, should be applied to the system development.

1. はじめに

現代社会において情報システムは欠かすことのできない社会基盤である．情報システムの不具合は直接的に我々の生活に影響を及ぼすことになる．情報システムを構築する者の責務を考えるとき，いかに品質の高いシステムを作るかという点は欠くことができない要素である．

メインフレームが中心的であった時代，情報システムのアーキテクチャはベンダーが独自に定義し，固有の技術体系が存在していた．そのためシステムの構成要素がベンダーの定義したアーキテクチャに準拠していることは当然であった．また固有の技術体系に関する知財はベンダー内に蓄積され，技術者間での知識共有が比較的容易であった．

一方，現代のオープン系の情報システムでは，アーキテクチャの自由度の高まりと同時に，情報システムの構成要素間に相互作用的な特性が生じた．技術者はこのような複雑性をも考慮しなくてはならなくなった．さらに現代においてはシステム開発技術の進化が早く技術者が習得すべき知識量が急速に増えている．開発期間の短縮化が求められる昨今の状況をあわせ考えると，情報システムを作るための知識・経験の蓄積及びその有効活用と共に，システム開発技術の変遷に影響されにくい基礎的，普遍的な部分を認識することが重要である．

本稿では，まずソフトウェア開発にまつわる課題について整理したうえで，技術の変化に耐えうる共通部分としてアーキテクトが備えるべき視点の意義を説明する．また，この視点を整理，体系化することによって享受できる効果について考察する．

2. ソフトウェア開発における課題

ソフトウェア開発を工学としてとらえ，品質や生産性を追及することは長年の課題である．1960 年代にソフトウェアエンジニアリングという言葉が生まれ[1]，それまで人間が行ってきた作業をコンピュータに行わせるために，システムの機能としていかにそれを実装するかということが考えられた．その後，ソフトウェア開発の生産性向上と開発期間の短縮，コストの削減が意識された．現代ではソフトウェアがさらに社会に浸透したため，重要な社会基盤を担うソフトウェアの品質をいかに高めるかということが重視されている．

これまでソフトウェアの品質を高めるために様々なアプローチがとられてきた．代表的な例として，開発標準や知識体系の活用がある．これらはソフトウェア開発者にとって認知度が高く，その有効性は広く認められている．以下において各々の効果と，それ単体では解決し難い課題について説明する．

2. 1 開発標準の効果と課題

ソフトウェアエンジニアリングの中でも開発標準は大きなテーマのひとつである．

TEAMmethod[*1] は米国ユニシス社で作成された手法，技法およびツールの組み合わせである．これは情報化計画，ビジネスプロセス再設計，情報システム設計，情報システム構築，プロジェクト管理など，チームを組んでシステムを開発する際の効果的な技術のアウトラインを示したものである．また TEAMmethod を基盤とした開発プロセスの標準技法である ISEP（Information Service Engineering Process）では，開発作業の手順及び各作業ステップの目的，作業内容，入出力情報が提示されている．

このような開発標準を導入することは，成果物品質を確保するために効果的な手段である．スキルや経験において多種多様な開発者が多数集まり，ひとつのシステムを作り上げる場合，開発標準は最低限の成果物品質を確保し各成果物間の整合性を保つための現実的なアプローチである．

注意点として，開発標準の適用には与えられたプロセスに従うこと自体が目的となってしまう危険性がある．開発標準に含まれる作業手順や作業内容がなぜそうなっているかという本質的な部分が開発者に意識されなくなる落とし穴に注意する必要がある．既存のプロセスをそのままあてはめることができない状況では，何のためにどのような作業をどのような手順で行うかについて考える必要がある．この際，固定化されたプロセスやテンプレートに従うことが常習化していると考慮に漏れが生じる可能性がある．

2. 2 知識体系の効果と課題

IEEE が中心となってまとめ，2001 年に制定されたソフトウェアエンジニアリングに関する知識体系として SWEBOK[2]がある．このドキュメントはソフトウェアに関する要求，設計，構築，テスト，保守などソフトウェアエンジニアリングに関する知識を整理したものとして世界中に認知されている．SWEBOK はソフトウェア開発においてすぐに活用できる知識や技術を詳細に解説してくれるものではなく，知識体系へのガイドである．SWEBOK の価値とは，バラバラに存在するソフトウェアエンジニアリングの知識を体系化し，一般に認められた知識へのインデックスを示していることである．SWEBOK には，キーワードに紐付けられた参照先を辿って多くの推奨文献にアクセスできるメリットがある．これによる効果として，ソフト

ウェア開発にどのような知識が必要であるかを俯瞰できることや，技術者間のコミュニケーションが容易になることがあげられる．

ただし，知識体系は名前の通り知識を対象としたものであるため普遍的なものではなく，時代とともに進化する技術や開発手法を取り込んでいく必要がある．現代においては新しい技術が次々と生まれるため，知識体系は技術の進化に追随できなくなる可能性がある．

上記で説明した開発標準や知識体系では重点が置かれていない部分に対して，「視点」が補完的な役割を果すと我々は考えている．次章からはその根拠と「視点の体系」とは何かについて解説する．

3.　アーキテクト視点の体系化の狙い

3. 1　普遍的な着眼点の意義

前章で述べたように，開発標準や知識体系の活用はソフトウェア品質向上のための手段として有効ではあるが，カバーできない領域も存在する．作業の根底にある本質的な部分が見えにくいこと，次々に登場する新しい技術のために知識が陳腐化することという二つの課題に対して「視点」というアプローチが有効であると考える．

我々が注目した「視点」とは，情報システムを捉える際に持つべき視点である．「視点」の特徴として大きく 2 点ある．

1 点目は多様性を吸収できることである．これは顧客が持つ多種多様なニーズへの対応が柔軟にできることを意味する．顧客のニーズは千差万別であるうえ，システム化する際の前提条件もそれぞれ異なる．さらに，時代と共に顧客ビジネスの内容や環境も変化していくため，情報システムの作り手は様々な関心事に対応しなくてはならない．

2 点目は技術変化の影響を受けにくいことである．顧客のニーズを認識できても，情報システムとして具現化できなければ意味が無い．顧客ニーズを要件化し，設計し，情報システムの形にできて初めて顧客ニーズを満たすことができる．情報システムをつくる技術の低さや，新技術への対応の遅れは，顧客ニーズを情報システムとして具現化する機会を損失することになる．進化する技術への追随は情報システムの作り手として重要であるが，同時に普遍的に変わらない考え方・着眼点を持ち続けることもまた重要である．

上記の理由から，日本ユニシスでは技術の進化や多様性に影響されにくい土台として「視点」に注目し，その知財化に取り組んだ．

3. 2　アーキテクトの役割と視点

前節で解説のとおり，ソフトウェア開発時に必要な普遍的な部分として視点があり，この視点は開発標準や知識体系では満たせない部分を補うことができる．

日本ユニシスでは，情報システムを作る上で必要とされる役割の内，アーキテクトに注目し，アーキテクトが情報システムのアーキテクチャを策定するにあたり本来備え持つべき視点を整理・体系化した（以下，「視点体系」）．アーキテクトは顧客の持つニーズを情報システムとしていかに実現するかを描く役割であり，情報システムの品質への影響が最も大きい役割といえる．

視点体系は主に要件定義時から設計時までのアーキテクトの視点を表している．視点体系は

純粋な視点を示しているのみであり，それに対する解法や具体的な技術を示しているわけではない．すなわち，この視点体系ではいわゆる「HOW」を表す記述を含めないようにしている．視点という抽象度を逸脱すると変化や多様性の影響を受けることになり，視点体系の存在価値が低減してしまう．

次章では情報システムのアーキテクチャとは何か，またアーキテクチャを形作る上でどのようなアーキテクト視点が必要かについて説明する．

4. 視点体系

4. 1 情報システムのアーキテクチャ

情報システムはハードウェア，ソフトウェアを含む様々な部品により構成され，それぞれが連携しながらその目的を達成している．アーキテクチャはそれらの部品（アーキテクチャ要素：4. 1. 1 項にて説明）の構造と特性，相互作用を表したものである．

システムの構造はデザイン時に表現される静的構造と，稼働時の状態を表現する動的構造の二つがある．静的構造は情報システムのデザイン時においてどのようなアーキテクチャ要素が存在し，どのような役割を果たし，互いにどのような関係を持つのかを定義するものである．たとえば，包含関係や，階層関係などである．それに対して動的構造はシステムの稼働時に情報システムに何が起こり，それに対して内的・外的にアーキテクチャ要素がどのように反応するかを定義するものである．あるイベントをトリガとして生成，更新，消滅などが起こり，状態が遷移することなどがこれにあたる．

システムの外的特性は，システムが何を（外的振舞い）どのように（品質特性）行うかを表している．外的振舞いは，システムが外部環境との間でどのように振舞い，どのような情報をやり取りするかという機能的な相互作用のことである．品質特性は非機能的な側面を表している．システムがどのように振舞うかを外部の観点から見たものである．

4. 1. 1 アーキテクチャ要素

アーキテクチャ要素とは，アーキテクチャを構成する部品であり，それらの責務，境界，インタフェースを明確に定義することがアーキテクチャ策定の必須条件となる．図 1 に示すように，アーキテクチャ要素は情報システムの構成要素であり，アーキテクチャの基になる．

アーキテクチャ要素はその抽象度を変えても，定義すべき項目は変わらない．つまり，アーキテクチャ要素の抽象度を徐々に下げることで詳細なアーキテクチャが策定される．また，ここで述べるアーキテクチャ要素にはアプリケーションのコンポーネントやモジュール以外にも，ミドルウェアやハードウェアなども含まれる．たとえばクラス，モジュール，サービスなどのソフトウェア要素や，リレーショナルデータベーステーブルやデータファイルなどのデータ要素，HDD，CPU，ルータなどのハードウェア要素がある．それぞれのアーキテクチャ要素について果たすべき責務を明確にし，それに基づき，アーキテクチャ要素が持つべき機能と特性に関して明確な境界を定めることができる．

アーキテクチャ要素が他のアーキテクチャ要素に対してどのようなサービスを提供するかを，インタフェースの定義を行うことにより表現する．また，そのアーキテクチャ要素が外部からどのような情報を受け取り，どのような情報を返すのかを定義することで，外部との関係が明確になる．

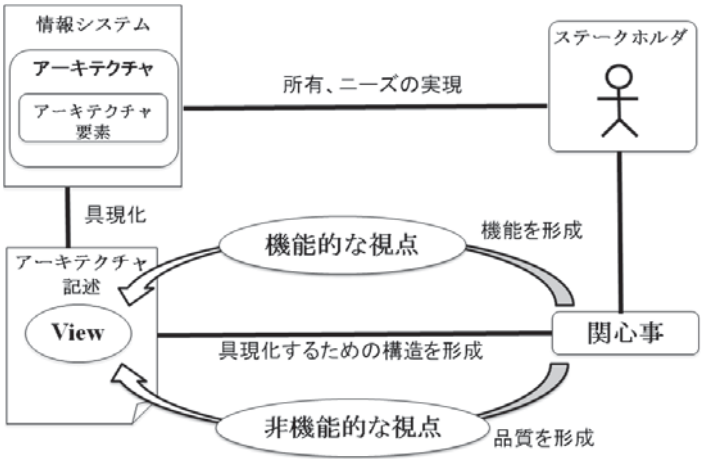

図1　ソフトウェア開発のモデル

4.1.2　ステークホルダと関心事

アーキテクチャの策定にあたり，様々なステークホルダの興味や関心事を満たすことを考えなければならない．また，場合によっては異なるステークホルダ間での対立する興味や関心事を調整する必要がある．ステークホルダはエンドユーザやシステムオーナといった顧客だけではなく，開発者，テスター，オペレータといったシステム側のグループ，さらにはエンティティといった非人物も含まれる．このことはシステムに影響を与える，もしくはシステムから影響を受けるすべての対象がステークホルダということを意味する．

様々なステークホルダが持つ興味や関心は，情報システムに対する要求や意見，目的とほぼ等しい．そのため，これらの関心事をアーキテクチャ要素で実現していくようにマッピングする必要がある．

4.2　視点の体系化

ステークホルダの関心事を情報システムとして具現化するために，アーキテクチャ記述の形で表現するのが一般的である．関心事を洗い出す際には機能的な側面と，非機能的な側面について十分に考慮する必要がある．もし，このとき漏れが発生すれば，完成する情報システムがステークホルダのニーズを満たさない可能性が生じる．しかし，関心事は表面化しているものもあれば，潜在的なものもあり網羅することは容易ではない．

たとえば，開発対象の情報システムにおいて高い可用性が求められているとき，アーキテクトは，高可用性を実現するためにどのような手段があるのかについて考えるはずである．しかし，手段の根底には次のような視点が必要となるのではないか．障害の発生率低減，障害の早期発見，障害の回復時間の短縮等である．この障害の早期発見に関しては，障害を認知するまでの時間を短縮する視点もあれば，障害状況の認知程度をより正確で詳細なものにするという視点も含まれている．さらに，障害の回復時間の短縮を考える際には，障害解析時間の短縮，復旧に要する時間の短縮，修復がシステムの他の部分へ影響を与えていないことを確認するための時間ということまで考える必要があるかもしれない．

経験豊富な技術者であればこのような視点を網羅することが可能かもしれないが，考慮漏れ

が発生するリスクは熟練した技術者においても皆無とは言えないはずである．本節では，アーキテクトとしてどのような視点があるか，その概要を説明する．視点体系は「機能的な視点」と「非機能的な視点」から構成され，アーキテクトがステークホルダの関心事を洗い出し，品質を考慮しながらアーキテクチャ記述を作る際の指針となるものである．なお，これらの視点を洗い出す際に Nick Rozanski らの “Software Systems Architecture”[3] を中心に，多数の書籍[4]～[18] と開発経験者の意見を参考とした．

4. 2. 1 機能的視点

アーキテクトは，機能的視点を用いてステークホルダの関心を分離し，アーキテクチャの構造と振舞いを定義する．機能的視点は下記の六つの視点に大きく分類できる．

1） 機能

アーキテクチャ要素の個々のスコープを明確にし，各要素の責務と相互作用を明確にするための視点である．これはアーキテクチャ要素の凝集性を高めることにつながる．また，アーキテクチャ要素の内部構造を考えると共に，各要素と外部の間でのメッセージと制御フローを明確にし，それぞれのアーキテクチャ要素が持つインタフェースの定義を明確にする必要がある．

2） 情報

情報システムにて扱う情報を蓄積，操作，管理，配信するための視点である．データベースに永続化される情報に限らず，メモリ上に展開される情報，アーキテクチャ要素間で受け渡しされるメッセージ，一時ファイルなどの情報を含め，そのライフサイクルや一貫性，データ量を考える必要がある．

3） 並列処理

複数の処理を同時並行的に実行する際の状態遷移，同期，整合性についての視点である．プロセス間でデータを共有する場合には，共有する場所や通信の仕組み，情報の構造についても考慮する必要がある．また，タスクが失敗した場合の検知や回復についても意識しなくてはならない．

4） 開発

開発者の負荷を軽減するための考慮であり，モジュール構成や標準化，構成管理など開発についての視点である．高凝集で疎結合なモジュール構成によって複雑性を低減できる．また共通機能を抽出，定義することで品質の制御が容易になると共に，開発生産性を向上させることができる．またチーム開発においては設計，構築，テストの標準化についても考慮が必要となる．

5） 配置

ソフトウェアを効率的に稼働させるハードウェアやネットワークといったインフラストラクチャのアーキテクチャに関する視点である．必要となるハードウェアの種類，仕様，数量

やソフトウェアに関する要件について考える必要がある．また，これらの互換性や設置面積，電力，室温，距離等の物理的制約についても注意を払うべきである．

6）　運用

情報システムが稼働を開始した後の制御，管理，監視等についての視点である．稼働中の情報システムの何についてどの程度の精度と頻度で監視を行うかについての考慮が必要である．アップデートに際しては，既存データや実行中の要素の状態を把握する必要がある．また移行においては機能やデータを正確，効率的に移行するための考慮が必要である．その他バックアップやリストアについても十分な注意が必要である．

4. 2. 2　**非機能的視点**

アーキテクトは，機能的視点を補う形で横断的に品質を定義していく．この際に必要となる観点が非機能的視点である．非機能的視点は下記の十の視点に大きく分類できる．

1）　セキュリティ

脅威からの攻撃によりシステムが機能を停止しないように防衛し，被害を最小限にするための視点である．「どのような資産を」「何から」守る為に「何を」すべきかという判断を明確化する視点，すなわち保護対象の認識と，脅威分析とリスク分析のための視点が必要となる．情報システムには防御や影響を最小限に留める策を講じると共に，問題が発生した際には原因をできるだけ正確に把握し，素早く回復させるための仕組みが必要である．

2）　パフォーマンスとスケーラビリティ

情報システムがどの程度の処理まで対処が可能であるか，また限界を超えた際に情報システムがどのような振る舞いをするのかについての視点である．スケーラビリティはシステムの処理数が限界に達した際，システムのキャパシティを上げるための視点である．レスポンスタイムやターンアラウンドタイム，スループットに関連するアーキテクチャ要素の能力は適切か，リソースの割り振りは適切かという視点が必要である．また，ハードウェアの性能と共にコストに関する考慮を欠くことはできない．

3）　可用性と回復性

システムが稼働を停止している時間をいかに短くするかという視点である．つまり，障害が発生する頻度が少ないほど，また障害からの復旧が早いほど可用性が高いと言える．可用性のレベルを上げることは一般的にシステムの複雑性や開発，運用コストの増加を招くことになる．システムが備える可用性とそれによるコストのバランスを見極めなくてはならない．高い可用性が求められるシステムでは，縮退運転が可能な設計や障害原因を特定できない場合のコンティンジェンシープランなどの対策を検討しておくべきである．

4）　変更性

システムの変更の容易さについての視点である．システムに対する変更がシステム内外へ及ぼす影響を認識する必要がある．例えば，変更範囲，変更の難しさ，変更によるビジネス

への影響などである．変更の大きさは最終的に金銭的なコストに影響することに注意が必要である．変化への対応を素早く行うためには，情報システムの各要素の置換容易性，設定のパラメータ化，データと処理の分離，処理と表示の分離などの視点が必要となる．また，開発時にはシステムの「どこ」を「どのように」,「なぜ」変更したのかについて明記しておくことが望ましい．

5） アクセシビリティ

システムが利用者の能力レベルに適合しているかという視点である．利用者によって能力や特質が異なることを想定し，どのような範囲の利用者を対象とするのかについて検討する必要がある．例えば，身体的な機能障害や教育レベル，年齢や言語の違いについての思慮が要る．このような障壁を下げるために，システムには表示機能，発音機能，印字機能，入力補助機能などがある．当然ではあるが，障害者基本法や障害者支援法などの法律を遵守しなくてはならない．

6） 開発リソース

ソフトウェアの開発に必要な要素に関する視点である．十分な開発リソースを確保できないことは，開発する情報システムの機能や品質に対する制約になりうる．開発リソースには，開発期間や開発者の労働時間など時間的なリソース，システム構成要素や作業者にかけるための金銭的なリソース，知識やスキルを持った技術者など人的なリソースがある．

7） 国際化

システムを複数のロケールに対応させる為の視点である．例えば，国外でのサービス提供や，国内にいる他言語の利用者に対してサービスを提供する場合には，国際化の考慮が必要である．システムが特定のエンコーディングに依存しないような配慮や，システムが表示するメッセージやマニュアル等の言語についても国際化の視点が必要な場合がある．また，通貨などの単位，住所や日付などの表記方法，画面や紙のサイズを表す規格が異なる場合がある．

8） ロケーション

システムを配置する拠点やサービスを提供するエリアの環境に関する視点である．物理的に離れた拠点間の通信における効率性や信頼性，回復性などの特性についての考慮や，ネットワーク上を流れるデータの量に関しても把握しておく必要がある．また，電源や通信回線等のインフラ設備，気候や天災の特徴など国や地域独特の物理的な条件についての考慮も加味することで，その拠点に適したシステムを提供できるようになる．

9） 法律と規則

国が定めた法律，自治体が定めた条例，特定の業界や企業内で定めた規則についての視点である．システムの基本動作が法律に反しないように設計するとともに，事故または故意にかかわらず，情報システムの利用者が操作により違法行為を行うことを可能な限り防止するための考慮が必要である．また，データ保護やプライバシー情報の取り扱いに関する視点も，

法的なリスクや信頼失墜のリスクを回避するために必要となる．

10） ユーザビリティ

システムに接する人がシステムを効果的に利用できるための視点である．システムに接する人とは，必ずしもエンドユーザとは限らない．運用管理を実施する人やサポートに携わる人も含まれる．たとえシステムに備わっている機能が十分であっても，ユーザビリティが悪ければ利用者にとって機能がないことと同義である場合もある．ユーザインタフェースの見やすさ，分かり易さ，操作性，デザインなどの見た目以外にも，理解性や習得性のように利用者が目的を果たせるかどうかという視点もある．

4. 3 視点体系の適用

視点体系は前述のとおり特定の開発方法や技術に依存しない，アーキテクトに必要な普遍的な事項をまとめたものであり，開発方法やアーキテクチャの自由度，多様性を阻害しないことを前提としている．従って，特定の開発方法や開発フレームワークを否定するものではない．ただし，視点体系に記述されるポイントが考慮されていない開発方法や開発フレームワークは，考慮漏れが発生している可能性がある．視点体系と既存の開発方法や開発フレームワークを合わせて使用することにより，相互補完的にアーキテクチャ策定における品質を向上させることができる．視点の記述は抽象的なものになるため，単体ではなく実装技術や開発プロセスと組み合わせ，顧客のニーズを満たす情報システムという形に具現化できてこそ存在価値がある．

視点体系を実プロジェクトにて活用する場合，留意すべきことがある．視点体系は多くの視点が体系的に記述されている．個々に見ると欠かせない様に思える視点であっても，システム全体としてみた場合に他の視点とトレードオフの関係になっている場合があるので注意が必要である．システム全体での整合性や一貫性を考えることはシステムを作る側の責任である．システムが意味を為すために必要不可欠な視点か否かという絶対的な基準，また他の視点に比べて優先度が高いか否かという相対的な優先順位についても意識する必要がある．

5. 今後の展望

視点体系を活用するための計画が日本ユニシスで進行中である．適用の場面として以下のようなものを想定している．

・「アーキテクトがプロジェクト内で作業を進める際のリファレンス」

システム開発の各段階において参照し，作業の漏れを発見したり考えを整理する為に使う．またプロジェクト初期から参照し，チーム内の語彙・認識の統一を図る．

・「作業やプロセスの意味について理解を深めるための補助」

アーキテクトとして必要な知識の不足を補うための教材として利用する．またアーキテクトに限らず，ある作業の意味や根拠が何であるかの気づきを得るために参照する．

・「ツールや方法論を評価する際の基準」

あるツールや手法において何が考慮されていて何が考慮されていないかを，視点体系と照らし合わせて明らかにする．それにより対象プロジェクトに特定のツールやプロセスが適しているかどうかを判断する．また既存の方法論を改良する場合や，新しい方法論を作る場面で考慮

漏れを低減するために参照する．

・「レビュ時のチェック項目」

第三者レビュ時のチェックシートとして視点体系に記述された項目を使用する．要件定義終了時，設計終了時において検討項目に考慮漏れがないかを確認する．また，要件定義から設計フェーズへの引き継ぎ時のチェックに利用する．

上記で述べた活用方法は，いずれも視点という抽象的な概念の利用に留まっている．しかし，実際の開発現場においては具体的な開発技術やスキルや道具が重要な役割を果たすことになる．今後は，抽象的な概念である視点を基軸に技術等の知財の蓄積，活用を検討する必要がある．またアーキテクト以外のロールに関する視点の体系化についても検討の余地がある．

6. **お わ り に**

現在，日本ユニシスでは視点体系をソフトウェア開発プロジェクトに試行適用し始めたところである．有効性の検証については順次実施していく予定である．ただし視点体系について短期間での客観的な評価は難しいため，今後の課題としたい．

＊1　TEAMmethod とは，米国ユニシス社のインフォメーション・サービス部門がサービス提供に使用する，手法，技法およびツールの組み合わせであり，以下の五つから構成されている．
TEAMmethod/Information Planning（情報化計画）
TEAMmethod/Business Process Redesign（ビジネスプロセス再設計-BPR-）
TEAMmethod/Design（情報システム設計）
TEAMmethod/Implement（情報システム構築）
TEAMmethod/プロジェクトマネジメント（プロジェクト管理）

参考文献

［1］ Mcllroy M. D., “Mass produced software components”, In Proceedings of Report on a Software Engineering Conference Sponsored by the NATO Science Committee, NATO Science Committee, 1968
［2］ IEEE，松本吉弘訳，“ソフトウェアエンジニアリング基礎知識体系―SWEBOK2004”，オーム社，2005
［3］ Nick Rozanski, Eoin Woods, “Software Systems Architecture”, Addison-Wesley Pearson Education, 2005
［4］ Roger S. Pressman，西康晴，榊原彰，内藤裕史訳，“実践ソフトウェアエンジニアリング―ソフトウェアプロフェッショナルのための基本知識―”，日科技連出版社，2005
［5］ Steve McConnell，松原友夫，山浦恒央訳，“ソフトウェア開発プロフェッショナル”，日経 BP 社，2005
［6］ Len Bass，Paul Clements，Rick Kazman，前田卓雄，加藤滋郎，吉野圭一，佐々木明博，新田修一訳，“実戦ソフトウェアアーキテクチャ”，日刊工業新聞社，2005
［7］ Michael Jackson，大野徇郎，山崎利治訳，“システム開発― JSD 法”，共立出版，1989
［8］ Karl E. Wiegers，滝沢徹，牧野祐子訳，“ソフトウェア開発の持つべき文化 IT Architects’ Archive ソフトウェア開発の課題”，翔泳社，2005
［9］ Frederick P.Brooks, Jr., 滝沢徹，牧野祐子，富澤昇訳，“人月の神話”，ピアソン エデュケーション，2002
［10］ Andrew Hunt，David Thomas，村上雅章訳，“達人プログラマー―システム開発の職人から名匠への道”，ピアソン エデュケーション，2000
［11］ Gerald Jay Sussman，Julie Sussman，Harold Abelson，和田英一訳，“計算機プログラムの構造と解釈”，ピアソン エデュケーション，2000
［12］ Joel Spolsky，青木靖訳，“Joel on Software”，オーム社，2005

［13］ Suzanne Robertson，James Robert，苅部英司訳，“ 要件プロセス完全修得法 ”，三元社，2002
［14］ Michael Jackson，玉井哲雄，酒匂寛訳，“ ソフトウェア要求と仕様―実践，原理，偏見の辞典 新紀元社情報工学シリーズ ”，新紀元社，2004
［15］ Karl E. Wiegers，渡部洋子訳，“ ソフトウェア要求 ”，日経 BP 社，2003
［16］ Capers Jones，富野寿訳，“ ソフトウェア品質のガイドライン ”，構造計画研究所，1999
［17］ 保田勝通，“ ソフトウェア品質保証の考え方と実際―オープン化時代に向けての体系的アプローチ ”，日科技連出版社，1995
［18］ John McGarry，Cheryl Jones，Elizabeth Clark，David Card，Beth Layman，古山恒夫，富野寿訳，“ 実践的ソフトウェア測定 ”，構造計画研究所，2004

執筆者紹介　千 葉　浩 太 郎（Kotaro Chiba）
2003 年東京理科大学大学院　修士課程修了（工学）．同年日本ユニシス（株）入社．Java を用いた WEB アプリケーションの開発および，アプリケーションフレームワーク製品 LWF 主管業務に従事．現在，総合技術研究所 OSS センター企画推進室に所属．