高品質なプロジェクトマネジメントを実現するトレーサビリティ・マトリックスの構築

―　プロセス中心から情報中心のプロジェクトマネジメントへの変革に向けた基礎理論の提案　―

榮谷 昭宏[1,2]*、狼 嘉彰[1]、神武 直彦[1]

ソフトウエア開発プロジェクトにおいて、設計に関する情報は予算や品質を左右する重要なものである。そこで、この研究ではプロセス中心のプロジェクトマネジメント技術では見え難い情報に着目し、情報がプロジェクト内でどのように移転していくのかトレースを可能とするモデルを構築した。そのモデルにより、トレースの複雑性を定量化するトレーサビリティ・マトリックス手法を構築した。そして、ソフトウエア開発プロジェクトにそのモデルと手法を適用することで高品質な情報中心のプロジェクトマネジメントを実現する手法を示した。

キーワード：トレーサビリティ・マトリックス、複雑性、定量化、情報中心、プロジェクトマネジメント

Construction of a traceability matrix for high quality project management

– A proposal of a basic theory toward a change from process-centric management to information-centric project management–

Akihiro SAKAEDANI[1,2]*, Yoshiaki OHKAMI[1] and Naohiko KOHTAKE[1]

Design information is important for software development projects because it determines their cost and product. In this research, a model has been made which can trace how information moves in a project by paying attention to the design information which is hard to trace by process-centric project management. On the basis of the model, a traceability matrix method has been constructed which quantifies the complexity of the traceability. It has been confirmed that high quality information-centric project management is realized by applying the model and the method to software development projects.

Keywords：Traceability matrix, complexity, quantitation, information centric, project management

1　はじめに

PMBOK[1] によると“プロジェクトとは独自のプロダクト、サービス、所産を創造するために実施する有機性の業務である。”と定義されている。独自の製品やサービス等を創造するためには、テクノロジー、ヒューマンリソース等さまざまな要素を構成していくことが必要である。つまり、プロジェクトとは構成そのものなのである。そのような構成学の対象となるプロジェクトを高品質にマネジメントするために、現在までも多くの取り組みがなされ先行研究も多い。Visualizing Project Management[2]、PMBOK はもとより、ソフトウエア開発では SWEBOK[3]、Rational Unified Process[4][5] 等がプロジェクトマネジメント手法、開発プロセス手法としてそのノウハウを整理している。しかし、このような手法が整理されているにも関わらず、特にソフトウエア開発においては、いまだにプロジェクトの成功率は向上していない。例えば日本情報システム・ユーザー協会のレポート[6] によると 2004 年から 2008 年まで、500 人月以上のプロジェクトでは予算超過案件が半数を占めており、その傾向は毎年一定している。また Frank[7] は上記と同様に Standish Group が指摘している「プロジェクトの現状として 68 % のプロジェクトが失敗」というレポート[8] を取り上げ、既存のマネジメント技術の有効性について問題提起をしている。具体的には、プロジェクトマネジメント、システムエンジニアリングに関する標準類（ISO15288, IEEE 1220, EIA632, CMMI, INCOSE Handbook, and PMBOK Guide 等）をレビューし、それらすべてがプロセス中心であることを指摘

1 慶應義塾大学大学院システムデザイン・マネジメント研究科　〒 223-8526 横浜市港北区日吉 4-1-1、2 NTT コムウェア株式会社 〒 108-8019 港区港南 1-9-1 NTT 品川 TWINS アネックスビル
1. Graduate School of System Design and Management, Keio University　4-1-1 Hiyoshi, Kouhoku-ku, Yokohama 223-8526, Japan,
2. NTT COMWARE CORPORATION　NTT Shinagawa TWINS Annex Bldg.　1-9-1 Konan, Minato-ku, Tokyo 108-8019, Japan
＊ E-mail: a.sakaedani@sdm.keio.ac.jp

Original manuscript received July 26, 2011, Revisions received November 7, 2011, Accepted December 28, 2011

している。そして「既存のプロジェクトマネジメント技術、システムエンジニアリング技術は、さらにより良くマネジメントするための方法論を求めている」と述べ、既存のプロセス中心のマネジメントの限界を指摘している。

そこでこの研究では、より高品質なプロジェクトマネジメントを実現するための方法論を構築することを目的とし、その実現のために、プロジェクトの全体と細部を的確に把握し“木も見て森も見る”方法論を構築する。具体的な手段として、プロジェクトのアーキテクチャ（＝システムを構成する要素とその要素間の関係性を整理したもの）を明らかにする。そしてプロジェクトのアーキテクチャの複雑性を指標として全体を俯瞰し、また個々の要素の難しさ・要素間の関係性も指標化することで、プロジェクトの状況を定量的にマネジメントすることを可能にする（図 1）。最後に、そのアーキテクチャを示したプロジェクトのモデルを用いて高品質なプロジェクトマネジメントを実現する方法論を検討していくこととする。

2　プロジェクトマネジメントの現状とその問題点の分析

2.1　移転コストの観点から考える情報とモノの違い

移転コストの観点から情報とモノとは異なる特性をもつ。以降では、情報とは人のもつ知識やノウハウ、製品や文書等で形式化された知識やノウハウ等を指し、モノとは物理的に有形なモノと定義する。そのように考えると、例えばモノを受け渡して組み立てる自動車工場のような場合は、担当者がある作業を終え次の作業者に渡せば一つの作業は完了する。しかしソフトウエア開発のプロジェクト内では、情報が主たる移転物でありその移転コストにはモノと異なる特徴がある。情報移転コストとは、情報の探し手（受け手）に利用可能な形で情報を移転するためにかかる総費用と定義される [9]。情報移転コストをコントロールするには暗黙知の移転の難しさ、情報の受け手と送り手の能力差による移転の難しさ、情報量が多い場合の移転の難しさ

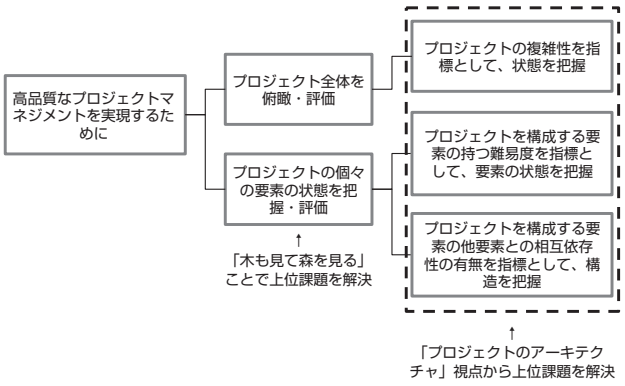


図 1　この研究の構成

表 1　情報とモノの移転コストの違い

移転コストの決定要因	情報	モノ
移転対象の性質	情報内容の暗黙性（ノウハウなど）により、その移転に大きなコストがかかる。	モノ自体の性質に依存して移転コストは変化しない。（移転速度や量など、移転条件に依存）
移転対象の送り手と受け手の属性	受け手と送り手の能力によって、情報の移転コストが変化する。	送受信者の属性には依存しない。（受け手から送り手に渡してしまえば、移転は完了）
移転されなければならない移転対象物の量	複雑で情報量が多いほど多くの情報交換が必要となり、移転コストが増加する。	量が多いと移転コストは増す。（量が多くても運んでしまえば 1 回で移転作業は完了）

を考慮しなければならない（表 1）。

ある作業者から移転される情報を次の作業者が元の情報を確認することなく、その情報の精度と詳細さを保つことは難しい。情報はモノとは異なり、モノを移転する場合のようにプロセスを明確に区切り、情報移転を進めることは難しく、その移転のために何度も同じプロセスを繰り返させる特性をもっている。

2.2　プロセス中心から情報中心のプロジェクトマネジメントの必要性

ソフトウエア開発のプロジェクトでは、物理的なモノではない情報の移転が主体となっている。つまり、情報中心であるソフトウエア開発プロジェクトの生産効率を向上させるには、決められた手順どおりの作業を円滑に進めることを追及したプロセス中心の改善だけでは限界がある。プロセス中心だけでは情報移転に関する特性（暗黙知の移転の難しさ、情報の受け手と送り手の能力差による移転の難しさ、情報量が多い場合の移転の難しさ）から生じる問題は解決しない。プロセス中心から情報中心への変革が必要なのである。

2.3　情報の特性：多義性と情報粘着性

情報中心で考える上で、情報移転コストについて再考する。そこで、情報に関する以下の二つの概念に着目した。以降では情報移転コストとこの二つの概念の関係を説明する。

情報粘着性：情報移転の難しさ [9]

多義性：情報が多様な意味に解釈されてしまう性質

情報粘着性とは情報を移転する場合、情報のもつ性質である粘着性のため、送り手から受け手に対して情報を移転することの難しさを意味する用語である。粘着性とは要素とそれがもつ情報が不可分なことを指し、この粘着性が原因で情報移転コストが生じる。粘着性の要因としては表 1 に上げたような 3 つの決定要因がある。情報の送り手と受け手が 1：1 の関係の場合はその 3 つの要因に起因する情報粘着性によって情報移転コストが規定される。例えば、暗黙知の形式化は多くの企業で推進されていることである

が、それは暗黙知のままでは情報粘着性が高くその情報移転コスト、つまり社内のある社員のもっているノウハウを他の社員に習熟させるためのコストがとても大きくなってしまう。ノウハウを引き出す、または学ぶために何度も情報のやり取りをする必要が生じることが考えられる。またはそもそも他の社員に移転することすらできないかもしれない。その結果、企業全体の能力を底上げするための障壁になってしまう。そのため、社員のもっているノウハウを形式化（見える化）することで、情報をやり取りする回数を削減し、情報移転コストの低減を図ることが必要なのである。

一方、プロジェクト等組織の中の1：nの関係において、情報移転コストを説明するには情報粘着性だけでは不十分であると考えた。なぜならば、送り手が別々ならば、受け手との関係も一意に決まり、その送り手のもつ情報粘着性を合計することで情報移転コストは得られる。しかし、送り手が同じ場合は、送り手のもつ情報粘着性を個々の受け手に合わせるためのコストも加味しなければならない。言い換えれば、ある受け手から別の受け手に情報移転先を変えるときに準備を行うためのコストを考慮しなければならないと考えたからである。そのコストを加味するために多義性という概念の導入を考える。

多義性とはある情報は受け手の観点により多くの意味をもってしまうことを言う。例えば一人の受け手に情報を移転するときに一つの情報が意味A、Bの2種類の意味付けがなされるかもしれない。しかし、複数人の場合はその意味付けがA、Bだけでなく、他の人からはC、Dとして捉えられるかもしれない。そのように情報移転先が増えれば、その分多義性も増す。情報の送り手は、複数の受け手に正しい意味付けをしてもらうために、例えば説明用の資料に複数の受け手からも正しく理解されるような情報をあらかじめ盛り込んだり、または受け手に応じて説明用資料を作り直す等を行う。受け手が特定され情報移転が始まれば情報粘着性の問題で処理できるが、受け手が特定されなかったり、情報移転が始まる前には、情報粘着性ではなくこのような多義性による移転コストを考慮する必要があると考えた。以上のように、1：nの関係で生じる多義性によるコストは、その受け手の数に依存している。多義性を抑えるためには受け手の数も低減することが必要なのである。また次に、n：1の場合は移転されたn個の多義的な情報を統合して、一つの意味付けにするという思考の整理が必要になる。n個の情報の意味について整合性をとり、一つの意味をもった情報として統合するのである。その統合作業もnに依存して情報移転コストも変わってくると考えられ、1：nと同様にnの数は少なくすることが望ましい。

さらに、限られた予算や時間のもとで行うプロジェクトでは、情報移転コストが高くなればその制約上、情報移転が完全に終わらないまま作業を進めてしまうケースが考えられる。そのような状況を避けるためには、極力移転コストを下げた状態をプロジェクト内に作り出し、それをマネジメントしていくことが必要になってくる。したがって、情報中心であるソフトウエア開発プロジェクトにおいては、情報粘着性と多義性をマネジメントすることが、送り手と受け手でやり取りされる情報の正確な移転を促し、より高品質なプロジェクトマネジメントを実現する上で重要であると考えた。

3　プロジェクトのアーキテクチャ（トレーサビリティ・マトリックス）の構築

情報中心のプロジェクトにおいて、情報粘着性と多義性をマネジメントするには、プロジェクトを構成する要素にどのようなものがあり、それぞれの要素がどのような関係性にあるのか整理する必要がある。そのためのモデルを構築する。

3.1　構成要素の抽出を実現する概念

オブジェクト指向によるビジネスモデリングにより要素を抽出する。特にオブジェクト指向を概念に取り入れた要求工学[10]によるとプロジェクトというシステムを構成する要素として、ニーズ（Needs）[用語1]、基本要件（Feature）[用語2]、要求条件（Requirement）[用語3]が抽出される。ニーズは基本要件と、基本要件は要求条件と関係する。機能(Function)[用語4]、実体または部品（Component）[用語5]は「全てのものは機能をもつ」[11]により、要素として抽出される。部品は機能と関係する。そして、プロセスフローから、成果物(Artifact)[用語6]、作業（Activity）[用語7]、担当（Team）[用語8]を要素として抽出する。担当は作業と、作業は成果物と関係する。

3.2　要素間の関係性整理を実現する概念と関係性の事例

プロジェクトというシステム全体のアーキテクチャを定義する上で、機械工学の「公理的設計」[12][13]と、組織科学の「ビジネスアーキテクチャ」[14][15]という概念の二つに着目した。組織活動を設計する上ではEA（エンタープライズアーキテクチャ）が類似したものと考えられるが、EA手法では参照アーキテクチャ[用語9]までは示してはいない[16]。そのため、自分でシステムを構成する要素とその関係性を定義する必要がある。しかし、この二つの概念ではものづくりのプロセスからものづくりに携わる組織、そしてそもそもどのようなものを作るべきなのかという顧客要求までも考慮し、その要素間の関係性について指針まで示しているものであることに大きな特徴がある。

公理的設計の考え方は、顧客領域・機能領域・実体領域・プロセス領域の各領域間で情報が写像され設計活動

がなされるという概念である。例えば顧客領域にある要求条件という設計にかかわる情報が、機能領域における機能に対する仕様として写像され、機能を実現するという考え方である。他も同様に各領域を同じ情報が写像され、各領域で最適な形に翻訳されて処理されているのである。また、その情報の写像は双方向に発生すると考えられている。そして、この概念に 3.1 で抽出した要素を当てはめると次のようになる。顧客領域は顧客の要件等を指すことから、ニーズ、基本要件、要求条件という要素で構成される。機能領域は文字どおり、機能という要素で構成する。また実体領域は設計解、つまり製品自体を示すので、部品および実体という要素で構成される。そして、プロセス領域は生産条件を指すので、成果物、作業、担当という要素が構成することとなる。以上により、開発プロジェクトという組織システムのアーキテクチャが規定された。つまり、開発プロジェクトの構成要素とその関係性を明らかにすることができた。

3.3　情報移転のトレースを実現する概念

オブジェクト指向で分析し、抽出した要素の関係を、公理的設計の理論に基づいて関連性を整理すると図 2 のようになる。

要求を把握するために、はじめにニーズを把握し、それを基本要件として整理し（図 2 では簡略化のため、この 2 項目は省いている）、最後に要求条件として定義する。また要求条件を実現するための機能を実装している実体が必要であり、その実体を作るためには設計内容をまとめた成果物が存在する。そして、成果物を作るための開発作業は、各担当で実行される。

ソフトウエア開発プロジェクト内で写像される情報の移転は例えば以下のようなケースが挙げられる。「ユーザー ID（以下 UID と記述）とパスワード（以下 PW と記述）を入力時にそれぞれの入力桁数チェックする」機能を実現する場合、各機能をどのような部品にどう実装するか設計しなければならない。例えば UID と PW に対して、それぞれ個別に桁数チェックの機能を部品として実装するのか、それとも桁数チェックは一つの部品として実装し、個々の ID チェックは同じ部品を共用するように実装するのか、どちらかの構造が選択される。それにより部品に対する設計も当然異なる。また、桁数チェック機能を共通化するならば、UID の体系、PW の体系が明確になるまでそのチェック機能の開発に着手できない。しかし個別に実装するならば片方の ID 体系が明確になれば、その設計は着手可能である。このように機能や部品、作業という要素が設計情報を必要な形に翻訳し移転し、影響を及ぼしあっており、その情報の移転コストは、プロジェクトの品質にも影響を及ぼす。

次にその情報移転コストについて考える。図 2 の左図は各要素が直線的に相互依存しているのみであるが、右図ではメッシュ構造となっている。メッシュ構造の方が複数の要素と情報をやり取りするため、多義性の例で説明したように情報移転コストもかかると考える。また図 2 の右図のような場合、例えば作業という要素について、その難易度が高いと情報粘着性も高まるため情報移転コストが増加する。設計書の内容をまとめるにあたって独自の表記法を用いている場合、その表記法では設計情報として表現しきれない内容があるかもしれない。その結果、設計情報が漏

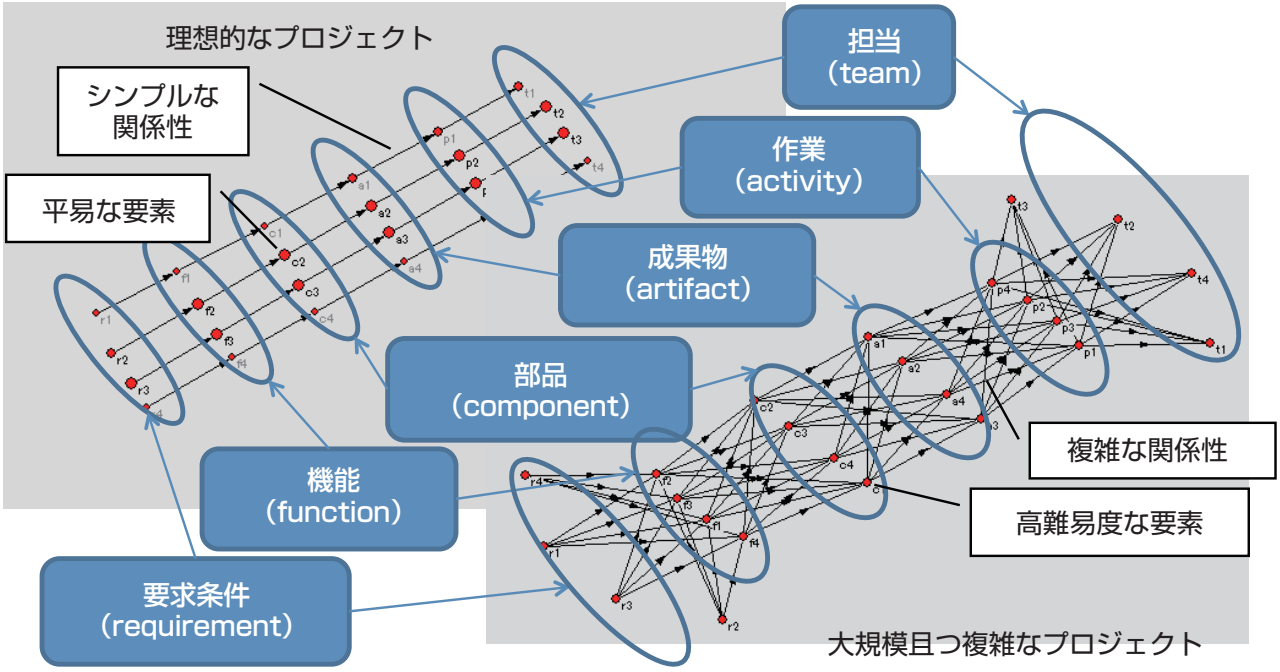


図 2　理想的なプロジェクトと大規模且つ複雑なプロジェクトのネットワークモデル

れ、設計ミスを誘発してしまう。独自の表記方法という情報移転を難しくする要因により、高い移転コストを生じてしまう事例である。したがって、要素間の関係は、極力独立性を保つこと、具体的には行列を用いて二つの要素間の関係を整理した場合、一番独立性の高い対角行列に近づけること、また要素自体の難易度（例えば前述のように作業の難しさを意味する概念。詳細は後述）を低減することが情報移転コストを低減するために必要である。言い換えると、プロジェクトのアーキクチャにおいて情報移転コストを低減し正確に情報を移転するためには、情報粘着性と多義性という概念を要素間の相互依存性と要素自体のもつ難易度で制御することが必要である。

3.4　プロジェクトのアーキテクチャ

基礎概念で示した図２の関係性をマトリックスで表現すると図３のようになる。

この関係性はベクトルを用いると以下のような行列計算（式１）と同意である。

$$\begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \begin{pmatrix} \text{Requirement-} \\ \text{Function} \end{pmatrix} \times \begin{pmatrix} \text{Function-} \\ \text{Component} \end{pmatrix} \times \begin{pmatrix} \text{Component-} \\ \text{Artifact} \end{pmatrix} \times \begin{pmatrix} \text{Artifact-} \\ \text{Activity} \end{pmatrix} \times \begin{pmatrix} \text{Activity-} \\ \text{Team} \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix}$$

$$\vec{r} = A\vec{t}$$

・・・・（式１）

$\vec{r}$：要求条件ベクトル、　　$\vec{t}$：チームベクトル、
A：システムマトリックス

その結果得られたトレーサビリティ・マトリックスはこの研究ではシステム全体の特性を示すという意味でシステムマトリックスと呼ぶこととする。このように行列を掛け算することで、担当と要求条件との関係を明示するトレーサビリティ・マトリックス（システムマトリックス）を得ることができる。

4　木も見て森も見るための指標

要素間で正確な情報移転を促し、より高品質なプロジェクトマネジメントを実現するためには、前述のとおり情報粘着性と多義性をマネジメントすることが必要である。多義性をマネジメントするためには、担当、作業、成果物等の領域間または領域内での要素間の関係性（以下、相互依存性と定義）を制御する必要がある。関係性のある要素数の増加に伴って情報移転を誤る確率は高まるからである。つまり、相互依存性が高まれば多義性も増加する。多義性により生じるコストを無理に低減してしまうと、複数の受け手が誤った意味付けを行う可能性も増加する。また、情報粘着性をマネジメントするには要素自体が抱える正確な情報移転を阻む要因（以下、難易度と定義）を制御する必要がある。要素自体の特性によっては、情報を移転する際に理解すべき本質的な情報までたどり着かず、情報移転を誤る確率が高まる。つまり、難易度が高まれば情報粘着性も高まり、剥ぎ取るべき情報を剥ぎ取りきれずに移転を終えてしまうことも考えられる。

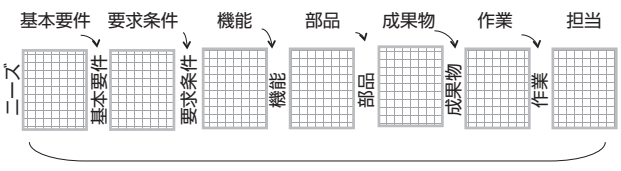


図３　プロジェクトのトレーサビリティ・マトリックス

以上から、情報移転の正確性は相互依存性と難易度で決まる。プロジェクト全体の視点から見ると、相互依存性を抑制して正しく情報移転する確率と難易度を下げて正しく情報移転する確率の積によって、プロジェクト全体で情報が正確に移転される確率も決まる。したがって、プロジェクト全体としては、相互依存性と難易度の積を複雑性と定義し、その複雑性を指標として制御する必要がある。

4.1　相互依存性の定量化

対角行列に近い方が全体の見通しも良くなる。また情報移転の観点からも多義性が低くなる。そのため、多義性の要因となる要素間の“相互依存性”の評価にはシステムマトリックスと単位行列との距離を測ればよい。単位行列と比較するにあたり、非対角成分の数が多いほど多義性が高くなることが表現できる必要がある。そのため、線形性のあるユークリッドノルムを用いて距離を測ることとした。つまり、評価するシステムマトリックスから単位行列を引き、そのマトリックスのユークリッドノルムが相互依存性となる。ただし、相互依存性に対するシステムマトリックスの成分値は、要素間の関係性がある場合は１を、関係性がない場合は０を設定することとする。以降はこれをシステムマトリックス s と呼ぶ。

4.2　難易度の定量化

要素の情報粘着性により生じる要素の“難易度”については、システムマトリックスの成分値で表現することとし、難易度を成分値とするシステムマトリックスをシステムマトリックス n と呼ぶ。このシステムマトリックス n 全体の大きさを評価することで難易度の定量化が可能になる。そのため、単位行列の何倍の大きさをもった行列であるか、つまりシステムマトリックス n のユークリッドノルムによって評価する。ただし、難易度に対するシステムマトリックス n の成

分値は、要素間の難易度を評価して決める（難易度の設定指針：参考資料1）。基準値は1であり、難易度が高いほど1より大きな値をとり、低ければ1より小さな値をとる。

4.3　複雑性の定義

前述のとおり情報移転の複雑性は以下のように定義する。

「複雑性＝難易度×相互依存性」

とする。ただし、難易度＝“システムマトリックス n”のユークリッドノルム、そして、相互依存性＝“システムマトリックス s － 単位行列”のユークリッドノルム、とする（計算事例：参考資料2）。

難易度・相互依存性は共にプロジェクトの個々の要素と要素間の状態を反映した変数である。したがって、それらの指標を捉えることで各行列の要素と要素間の関係性を捉えることができる。同時に、それを掛け合わせた複雑性の変化を捉えることでプロジェクト全体を捉えることができる。

4.4　正方行列化について

要素間の関係は必ずしも正方行列にはならない場合もある。その時には、相互依存性の算出に必要な対角化をするために正方行列化する必要が生じる。正方行列化するには成分値0を付与することで正方化する。成分値0であるので、正方行列化によって相互依存性は追加した次数分その値は変化する（なぜならば、正方行列化で追加した行または列の分の対角成分を引くため、追加分の対角成分が-1になるからである）。しかし完全な独立性は単位行列という正方行列で表されることを考えると、非正方行列における成分値0の付与による相互依存性の値の変化はその非正方行列の独立性を示す指標であると解釈される。以上のことから、成分値0の付与により正方行列化することをルールとする。

5　トレーサビリティ・マトリックスを活用したプロジェクトマネジメントのPDCAサイクル

この章ではプロジェクトの全体と細部を的確に把握し“木も見て森も見る”方法論を説明する。説明にあたってはPDCAサイクル（計画→実行→評価→改善のサイクル）をシナリオに用いる（図4）。

はじめに、トレーサビリティ・マトリックスを作成するために具体的に要素を整理し、マトリックスを構成する。そして出来上がったトレーサビリティ・マトリックスの難易度や相互依存性を改善してプロジェクト全体としての複雑性を低減する。そのプロジェクトで実際に開発作業を進め、その進捗状況をチェックする。進捗指標としてプロジェクトの複雑性を用い、その変化を見ることでプロジェクトの状況を俯瞰する。複雑性の変化が増加傾向にあるならば、それはプロジェクトの状況になんらかの問題が生じているため、その原因となっている要素を洗い出す。シミュレーションにより、要素の難易度や相互依存性を変化させ、プロジェクト全体の複雑性を低減することに一番効果的な要素を洗い出し、プロジェクトの状況を改善する。以降ではPDCAサイクルに沿って詳細に説明する。

5.1　PLAN（計画）

5.1.1　要素の洗い出しとその関係性の整理

図4に示したようにプロジェクトの現状分析を行い、システムマトリックスの作成に必要な要素の洗い出し、その関係性の整理を行う。しかし、要素の粒度がまちまち、または関係性が分からない等の理由からシステムマトリックスを作成できないことも想定される。そういった場合は、そもそもプロジェクト計画上になんらかの問題があると考えられるため、その解決を行う。

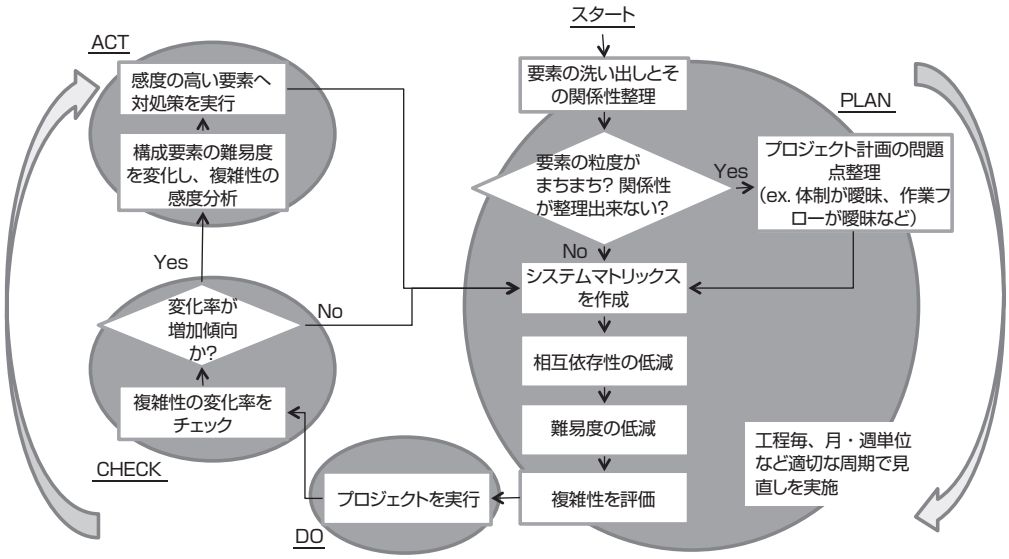


図4　プロジェクトマネジメントのPDCAサイクル

5.1.2　プロジェクト計画の問題整理

例えば体制が曖昧で他と比べて大きな粒度しか書けない担当があるならば、その整理を行うことが必要である。また他の例では、上流工程において要素を洗い出すとき、下流工程で明確になる部品や機能がはっきりしないことも想定される。その場合、見積もり根拠を確認する必要がある。見積もり時にはそれなりの根拠をもっているはずで、もし根拠がはっきりとしない場合は、プロジェクト計画立案時の課題として、早急な計画の再検証を行う必要ある。

以上を整理し、システムマトリックスｓとシステムマトリックスｎを作成する。

5.1.3　個々のマトリックスの相互依存性・難易度の低減と複雑性の評価

個々のマトリックスに対して対角化を目指した改善を行う。しかし、その結果、要素の難易度を上げてしまうケースが想定される。しかも、対角化した以外のマトリックスの要素の難易度を増大させてしまうケースも想定される。そのため、必ずプロジェクトを俯瞰的にみるために複雑性を算出してプロジェクト全体への影響度を確認することが必要である。

プロジェクトの構造を示すシステムマトリックスは図３で示したとおり、７つの行列の掛け算で導かれる。行列の掛け算の特性から、全行列または三角行列が組み込まれた計算の解は必ず全行列または三角行列となる。対角行列を解として得るには各行列を対角行列にする必要がある。

ステップ１として、システムマトリックスｓを構成する７つの各行列それぞれの関係性を整理し、対角行列に近づけることによって相互依存性を低減する。ステップ２として、システムマトリックスｎを構成する各行列で整理された関係性の中で成分値の高い関係、つまり難易度の高い関係を低減することによって、行列の難易度を低減する。以上によって、システムマトリックス全体の複雑性（＝相互依存性×難易度）の低減を実現する。

しかし、現実のプロジェクトではシステムマトリックスｓの７つすべての行列を対角化することは困難である。したがって、例えば各行列を三角行列化することを目的にプロジェクトの設計を改善する（相互依存性の低減）。そして、各要素間の関係性を例えば開発ツール等の導入によって平易なものとし、システムマトリックスｎの行列成分値の値を低減する（難易度の低減）。それにより全体としての複雑性を低減する案もある。また別案として、システムマトリックスｓの７つの行列の中で全行列は一つまたは二つの行列に絞り込み、他の５つか６つの行列は対角化し、システムマトリックスｎの成分値を極力低いものにするような改善も有効である。全行列である箇所の複雑性は高いかもしれないが、対角化した箇所の複雑性を低減することにより、全体の複雑性を下げるという考え方である。ただし、どちらの場合も必ずしも複雑性の低減に有効でないケースも有り得るため、プロジェクトの設計段階でのシミュレーションによる比較検証は必要である。

例えば、ソフトウエアをすべて手作りするスクラッチ開発[用語10]で計画を立てていたが、その複雑性を軽減するために、ソースコードを自動生成するツール（以下ジェネレータと記述）の利用、またはパッケージソフト（以下 PKG と記述）を利用するケースを検討する。その検討の流れを説明する（図 5）。スクラッチで開発をする場合の要素全体の関連は参考資料３に示した。特にその中で着目した要素のみを図６に切り出した。画面遷移部分のみ（いわゆる３層モデルのプレゼンテーション層）を対象としている。作業要素として画面設計、成果物要素として画面仕様書・画面遷移図、そしてそのソースコード、また部品としてログイン画面、メニュー画面等を、そして機能としては UID 入力機能、PW 入力機能等を挙げた。

このように要素の数が多く、相互依存性も高いことから、その改善を試みることとした。PKG では、その製品内に機能もすべて実装されていることから、設計者やプログラマーが個々の部品や機能を意識することがないと考え、部品として画面制御 PKG、機能として画面制御機能のみを要素として挙げた（図 7）。

また、ジェネレータは部品や機能の構造はスクラッチの場合と同じとした。しかし設計書からソースコードが自動生成されるので、設計者やプログラマーがソースコードを意識することは無い。そのためスクラッチとの構成要素の違いはソースコードの有無だけである（図 8）。

以上の３種類を比較評価した場合、一番相互依存性が低い PKG を選択することが適切と考えられた。しかし、その選択をした場合の周辺要素の影響を再考してみる。つまり、PKG に関する習熟度等を考慮したときに別の要素の難易度を増大させている可能性を考慮する。それぞれのシ

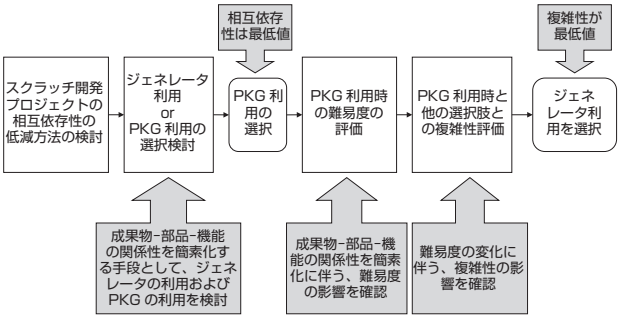


図５　相互依存性・難易度を考慮した複雑性の評価フロー

表 2　担当（team）－作業（activity）マトリックス

		担当（Team）				
		要求管理担当	アーキテクチャ設計担当	詳細設計担当	実装担当	テスト担当
作業（Activity）	画面設計	2.25	2.25	2.25	2.25	2.25
	業務ロジック設計	1.5	2.25	1.5	1.5	1.5
	DB 設計	1.5	2.25	1	1	1

表 3　各案の最終評価

	相互依存性	難易度	複雑性
スクラッチ開発	77.79	78.18	6081.42
ジェネレーター	64.40	64.75	4169.44
PKG 利用（未習熟時）	57.16	104.89	5995.12

ステムマトリックス n の成分値は基準となる 1 に設定していたが、この PKG の習熟度を考慮するのに伴い、表 2 のように成分値を定めた（機能と部品の関係等他のマトリックスの成分値は 1 のまま）。なお、表 2 では、担当（team）と作業（activity）の各要素に難易度（1 または 1.5）を設定し、マトリックスの成分値は行と列の難易度の掛け算によって設定した。

その結果、表 3 に示したようにプロジェクト全体として 104.89 の難易度を示すことが分かった。これはスクラッチ開発時の 78.18 という値を上回っており、また同時に他の選択案としてジェネレータを利用した場合の難易度(64.75) よりも大きい値を示した。

以上のデータを基に、PKG、ジェネレータおよびスクラッチのそれぞれを活用した場合のプロジェクトの複雑性を再評価した。その結果が表 3 であり、ジェネレータの選択が最適であると最終的に結論付けた。

5.2　DO（実行）

計画したとおりの形でプロジェクトを実行し、開発を進める。例えば上記の例ならばジェネレータを利用して開発を進める。

5.3　CHECK（評価）

前節まではある時点で複数のプロジェクトを比較した議論であったが、この節では一つのプロジェクトに閉じて、その時間的変化に対して議論を行う。

複雑性は開発が進むにつれて低減する。なぜならば、完全に情報移転が終わった要素間ではその相互依存性が無くなっていくためであり、最終的には相互依存性を示すマトリックスの要素値 =0 となる。同時に、習熟効果等により難易度も時間軸の変化に伴って低減する。ただし、難易度は必ずしも最終的に要素値 =0 になるとは限らない。例えば、限られた開発期間の中でパッケージソフトの全仕様を把握し、どのような場合でも対処可能なスキルを習熟することは通常は難しい。しかし、相互依存性は情報移転が完全に終われば 0 になるので、難易度を表すシステムマトリックスを Sn、システムマトリックスの次数をNとすると、システムマトリックス s から単位行列を引いた行列成分はすべて対角成分が -1 になる。したがって、そのノルムは$\sqrt{N}$になり、複雑性は以下の値に収束する（式 2）。

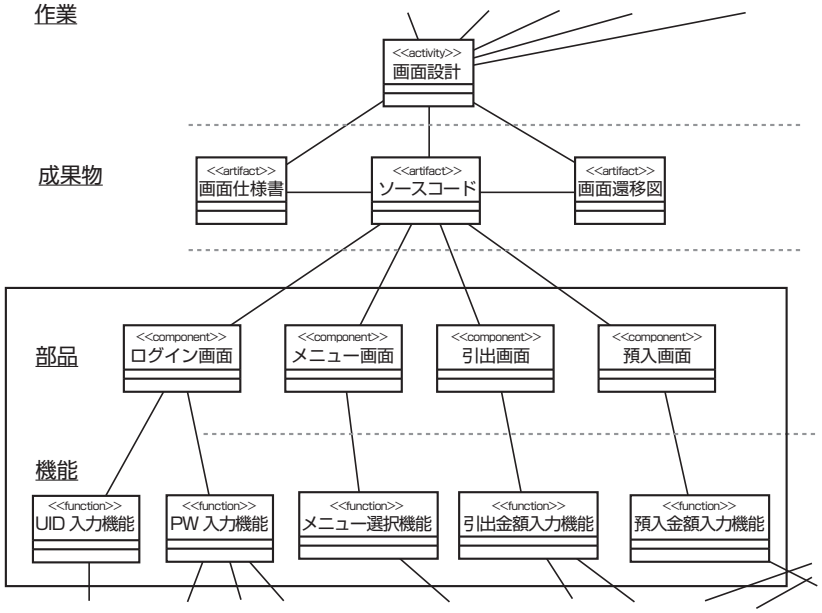


図 6　スクラッチ開発のネットワークモデル（画面遷移関連のみ）

$$\text{複雑性} = \sqrt{N} \times \|Sn\| \quad \cdots \text{(式 2)}$$

なお、完全に対角化されているシステムマトリックスは特異点と考え、式 2 の対象としない。完全に対角化されたシステムマトリックスとは図 2 における理想的なプロジェクトであり、そもそも複雑性を議論する対象にならないためである。

したがってプロジェクト全体の進捗状況を管理する方法として二つの観点が考えられる。

一つはプロジェクト全体を俯瞰する視点（森を見る）から、複雑性を管理指標として進捗管理を行う。複雑性の変化率が減少傾向にあるならば順調に開発が進行していることを示し、変化率が増加傾向にあるならばプロジェクトで何か問題が生じている可能性があることを示している（図 9）。

また、プロジェクトの個々の要素をマネジメントする観点（木を見る）からは、相互依存性を管理指標として進捗管理を行う。例えば、担当ごとに割り振られた要素を決め、その要素間の相互依存性を管理する。相互依存性の数とその変化率を進捗指標とすることも一つの案である。

5.4　ACT（改善）

次にプロジェクトの“木を見る”という観点から、先ほどの複雑性の変化率が増加傾向にあるならば、何が問題なのかをモデルから俯瞰し、その問題点を洗い出す。そして直接的にその問題を解決し、難易度や相互依存性を低減できるならば、その対処を行う。もし直接的に対処ができないならば、その他の要素によってシミュレーションを行い、その感度分析を行う（図 10）。この例では要素 a の方が要素 b よりも複雑性低減に効くことが分かる。

このように要素の難易度軽減によって複雑性の低減に効く要素を洗い出し、対処すべき要素を絞り込む。要素が決まれば、その要素の難易度低減に必要な対処を行う。

5.5　PDCAサイクル実行にあたっての留意点

プロジェクトマネジメント作業稼働の観点から PDCA 実行にあたって、以下の点に留意する必要がある。

まず、システムマトリックスの単位とマトリックスを構成する要素の粒度について検討する必要がある。プロジェクト全体を俯瞰するシステムマトリックス、またその中のグループやさらにその配下のサブグループでシステムマトリックス

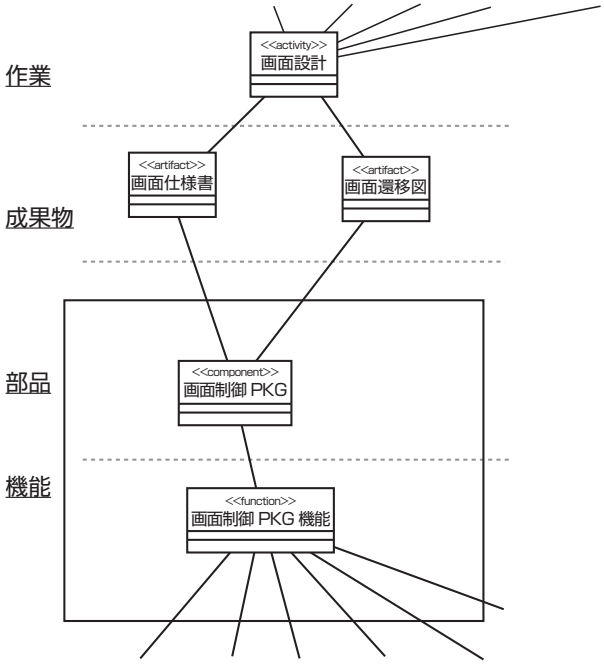


図 7　PKG 開発のネットワークモデル（画面遷移関連のみ）

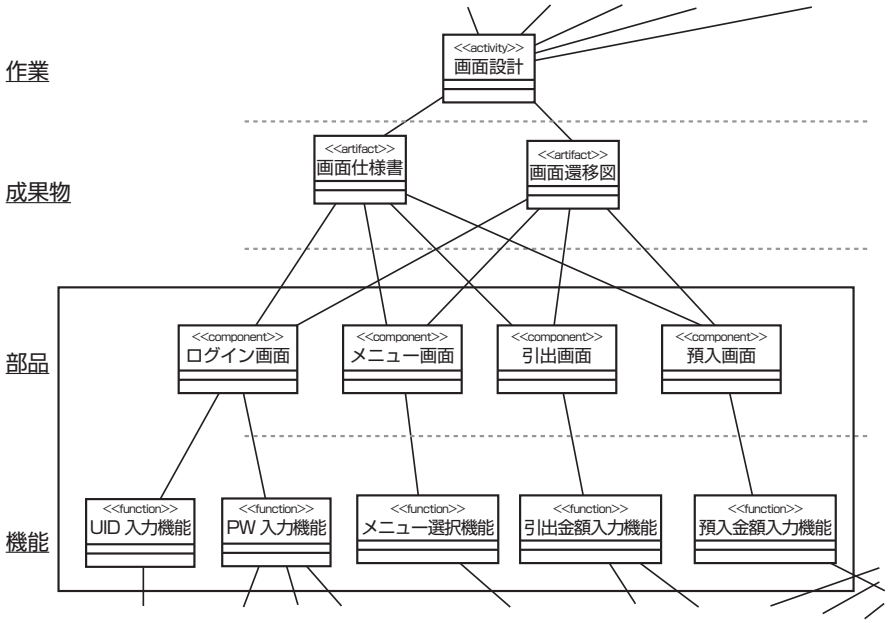


図 8　ジェネレータ開発のネットワークモデル(画面遷移関連のみ)

を作成するのか決める必要がある。そのマトリックス単位にあった要素の粒度を検討する必要がある。その構成要素が詳細過ぎると、マトリックスの維持管理のみで作業稼働が逼迫し、マネジメントがしきれない。そして次に、PDCA を回す周期を検討する必要がある。上記にも関連するが、システムマトリックスの単位または要素の粒度が大きいほど、その周期は長くすべきである。また粒度が小さければ周期は短くすべきである（図 11）。

具体的には担当や作業、成果物といった各領域単位に10から20程度の要素数に抑えられるようにシステムマトリックスの単位も検討すべきである。それ以上の要素数では、システムマトリックスはむろん、ネットワークモデルにおいても現状理解が困難で改善策も検討し難い。また、各領域の要素数を 10 から 20 程度にするにはマトリックスの作成単位だけでなく PDCA サイクルの周期も限定する必要がある。例えばサブグループレベルで全開発工程に対する要素を洗い出そうとしたら、その量は莫大であり 10 から 20 には到底納まらない。そのような観点からも周期を限定する必要がある。

以上を考慮することで、情報中心のプロジェクトマネジメントが実行可能となる。

6　結論

ソフトウエア開発プロジェクトのアーキテクチャとしてトレーサビリティ・マトリックスを構築した。そして、情報中心であるソフトウエア開発プロジェクトにおけるマネジメントの PDCA サイクルをシナリオとして、そのトレーサビリティ・マトリックスの活用方法を説明した。そして、プロジェクト全体を俯瞰的にかつ詳細に状況を把握できることを説明し、この手法が高品質なプロジェクトマネジメントを実現するための情報管理手法であることを示した。

謝辞

この研究は文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」の一環として実施された。この研究の概念に関してディスカッションを頂戴した慶應義塾大学大学院手嶋龍一教授、保井俊之特任教授に深く感謝申し上げる。

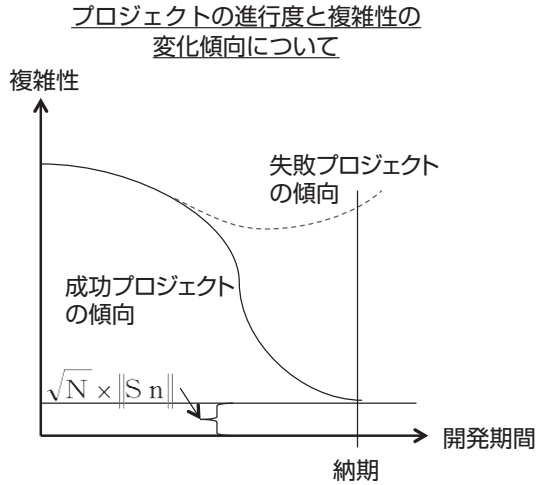


図 9　複雑性の継時変化とプロジェクト状態

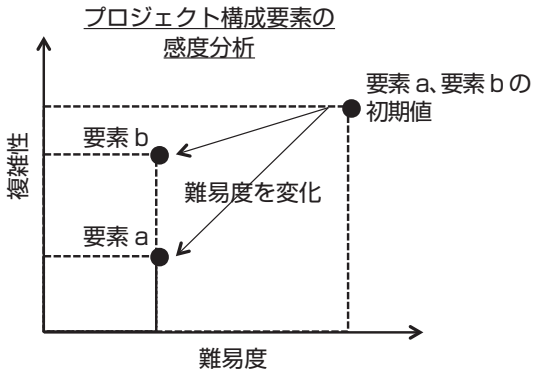


図 10　プロジェクトの複雑性のための感度分析例

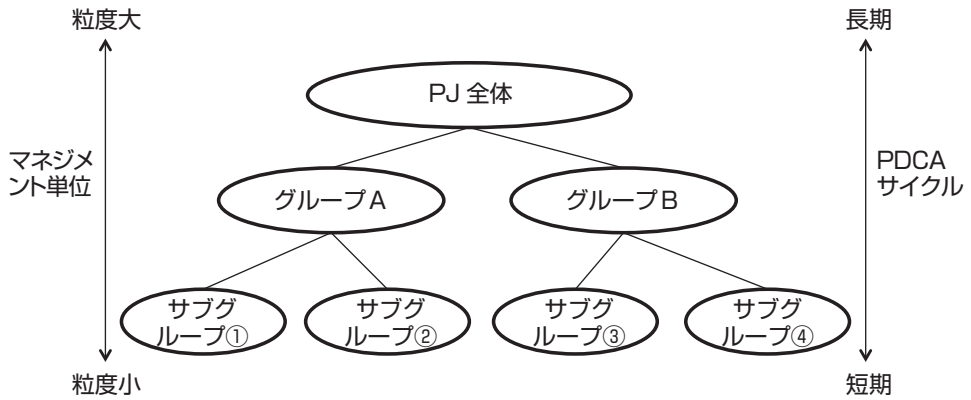


図 11　マネジメント単位と周期について

参考資料

参考資料1：難易度の設定指針

◆要素の「難易度」を“1”を基準として、以下の指標によりランク付けを行う。

項番	調査対象とする要素	測定対象	測定指標（例）	難易度ランク付け方針
1	組織（team）	各作業チームまたは担当者のスキル	経験年数、熟練度（スキル） 役割（リーダー? アシスタント? 等）	高スキル技術者（Team）→難易度低
2	作業（activity）	各作業の重要性	クリティカルポイントとなる作業か否か（補完的な作業か否か） 定型作業か否か	重要な作業（Activity）→難易度高 定型作業→難易度低
3	成果物（artifact）	各成果物の重要性	成果物作成にあたって特殊な技術が必要か否か（独自言語等の習得が必要等） 他の作業実施時に参照される数が多いか否か（重要な成果物なら参照数が多いと想定） 測定対象となる成果物のページ数、行数等が他と比較して多いのか否か 再利用性（他案件の成果物を再利用したものか否か）、またはプロトタイピング等によるベースの拡張度	重要な成果物（Artifact）→難易度高 （特殊技術要→難易度高、参照性高→難易度高、成果物の量が多い→難易度高、再利用性高→難易度低）
4	部品（component）	各部品の重要性（根幹となる機能、利用頻度等）	優先度の高い要求条件を実現するために欠かせない部品か否か 要求条件を実現するために、対象部品が利用される数が多いか否か（主要部品ならば他の部品とのインタフェース数は多いと想定） 再利用性（他案件の成果物を再利用したものか否か）、またはプロトタイピング等によるベースの拡張度	重要な部品（Component）→難易度高 （優先度高→難易度高、利用回数大→難易度高、再利用性高→難易度低）
5	機能（function）	各機能の重要性（根幹となる機能、利用頻度等）	優先度の高い要求条件を実現するために欠かせない機能か否か 要求条件を実現するために、対象機能が利用される数が多いか否か（主要機能ならば、稼動頻度も高いと想定） 再利用性（他案件の成果物を再利用したものか否か）、またはプロトタイピング等によるベースの拡張度	重要な機能（Function）→難易度高 （優先度高→難易度高、利用回数大→難易度高、再利用性高→難易度低）
6	要求条件（requirement） 基本要件（feature） ニーズ（needs）	要求条件の重要性、優先度	実現したい要求条件の優先度の高低	優先度の高い要求(Requirement)・基本要件（feature）・ニーズ(needs)→難易度高

・設定にあたっては、以下を留意すること。

基準は 1 とし、0.1 から 1.9 まで、同一カテゴリの調査対象とする要素の数に応じて細分化して値を付与することが必要である。例えば 10 個の要素がある場合に 3 種類程度しか難易度の値が無い場合、10 個の要素のもつ難易度の差が値を設定した時点で丸められてしまい、結果としてその差が見えにくくなる。むろん本当に難易度が同じという判断で 3 段階のみの難易度設定にすることは問題ない。また、0.1 から 1.9 としたのは、1 を基準とした場合に下限と上限を同じ幅とするためである。

参考資料2：複雑性の計算例

$(r1, r2)$ は要求条件を示すベクトルで、$(t1, t2)$ は担当を示すベクトルとする。

このシステムマトリックスで表される開発プロジェクトの複雑性を求める。相互依存性は関係の有無により行列の成分値を 0/1 のどちらかに設定する（システムマトリックス s）。難易度は個々の要素の難易度レベルにより数値設定し、行列成分値を決める（システムマトリックス n）。開発プロジェクト全体の複雑性は相互依存性と難易度を掛け合わせることによって算出されるのであるから、結果は以下のとおりとなる。

$$\begin{pmatrix} r1 \\ r2 \end{pmatrix} = \begin{pmatrix} \text{Requirement-} \\ \text{Function} \end{pmatrix} \times \begin{pmatrix} \text{Function-} \\ \text{Component} \end{pmatrix} \times \begin{pmatrix} \text{Component-} \\ \text{Artifact} \end{pmatrix} \times \begin{pmatrix} \text{Artifact-} \\ \text{Activity} \end{pmatrix} \times \begin{pmatrix} \text{Activity-} \\ \text{Team} \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \end{pmatrix}$$

$$= \begin{pmatrix} 1/2 & 1 \\ 1 & 1/2 \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \end{pmatrix}$$

難易度 = ‖ システムマトリックス n ‖

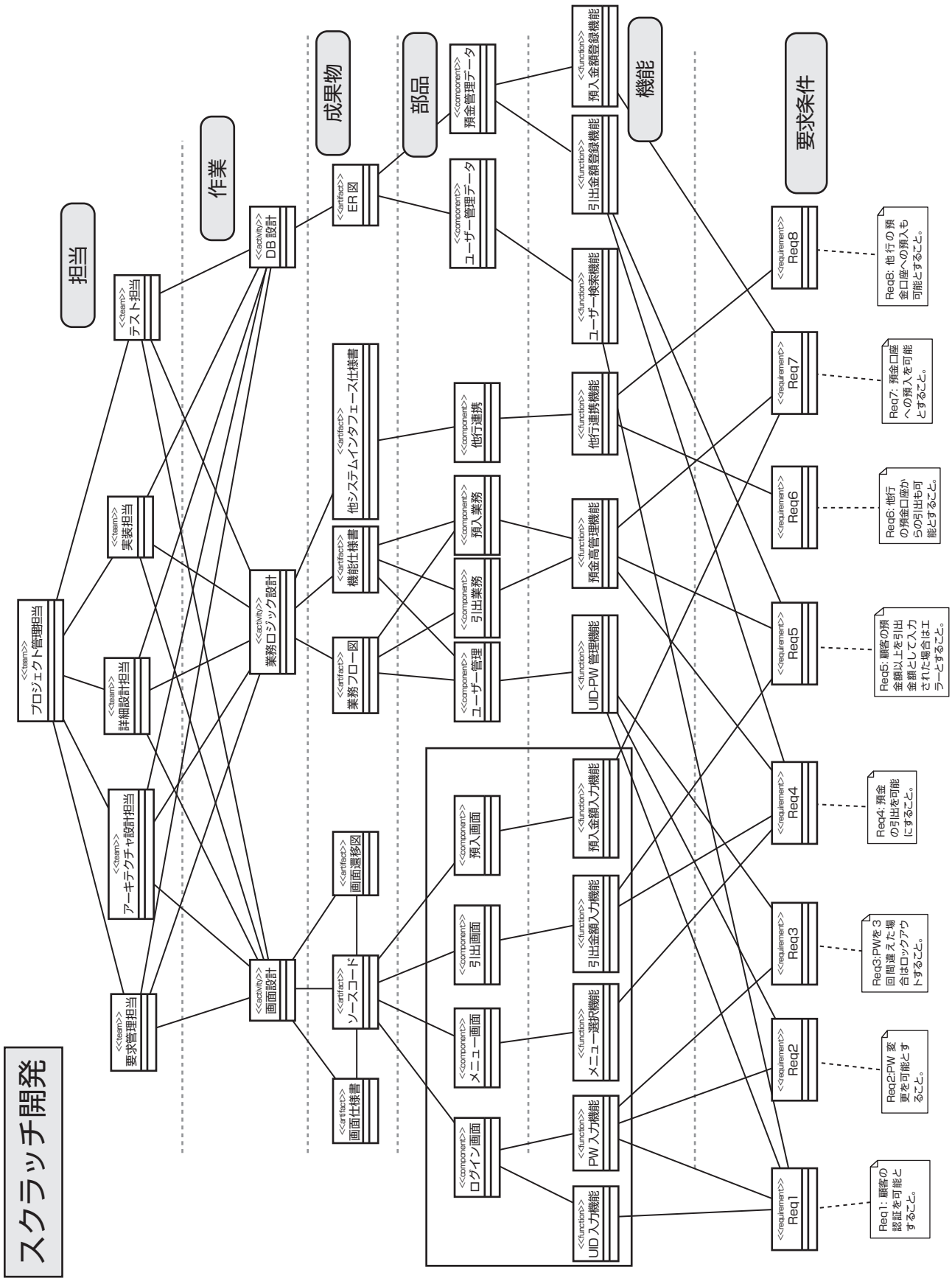
$$= \left\| \begin{pmatrix} 1/2 & 1 \\ 1 & 1/2 \end{pmatrix} \right\| = \sqrt{5/2}$$

相互依存性 = ‖ システムマトリックス s −単位行列 ‖

$$= \left\| \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right\| = \sqrt{2}$$

複雑性 $= \sqrt{5/2} \times \sqrt{2} = \sqrt{5}$

参考資料3：スクラッチ開発プロジェクトのネットワークモデル



用語説明

用語 1：ニーズ：Needs。例として、私達は○○がほしい。という表記方法が該当する。

用語 2：基本要件：Feature。例として：システムは××を実現すること。という表記方法が該当する。

用語 3：要求条件：Requirement。例として、システムにより△△を行い、□□を出力する。という表記方法が該当する。

用語 4：機能：Function。実体、部品の役割を示し、これにより要求条件を実現する。

用語 5：実体または部品：Component。成果物を実装したものであり、機能を有する。

用語 6：成果物：Artifact。作業により生産されるもの。例えば設計書等。

用語 7：作業：Activity。担当が実施する仕事を指し、成果物を生む。

用語 8：担当：Team。組織内で分担されるさまざまな役割ごとに配置されるチームまたは個人を指す。

用語 9：参照アーキテクチャ：ある領域に特化して作られたものであり、その領域のシステム設計を行ううえで参考にするアーキテクチャ。

用語 10：スクラッチ開発：開発者がすべてを実装する開発。

参考文献

[1] Project Management Institute, *PMBOKガイド 第4版*, Global Standard (2008).

[2] K. Forsberg, H. Mooz and H. Cotterman: *Visualizing Project Management Models and Frameworks for Mastering Complex Systems, Third Edition*, John Wiley&Sons (2005).

[3] *SWEBOK*, http://www.computer.org/portal/web/swebok, accessed in 2011

[4] P. Krucheten: *The Rational Unified Process: An Introduction, Second Edition*, Pearson Education (2001).

[5] W. Royce: *Software Project Management A Unified Framework*, Addison-Wesley (1998).

[6]（社）日本情報システム・ユーザー協会：*第15回企業IT動向調査2009*, http://www.juas.or.jp/servey/it09/summary09_0507.pdf , accessed in 2010

[7] M. Frank, A. Sadeh and S. Ashkenasi: The relationship among systems engineers' capacity for engineering systems thinking, project types, and project success, *Project Management Journal*, 42 (5), 31-41 (2011).

[8] Standish Group Chaos summary 2009, The ten laws of chaos, http://www1.standishgroup.com/newsroom/chaos_2009.php, accessed in 2011

[9] E. V. Hippel: "Sticky information" and the locus of the problem solving: Implications for innovation, *Management Science*, 40 (4), 429-439 (1994).

[10] Dean Leffingwell and Don Widrig: *Managing Software Requirements : A Unified Approach.*, Addison-Wesley (2000).

[11] 吉川弘之: サービス工学序説－サービスを理論的に扱うための枠組み－, *Synthesiology*, 1 (2), 111-122 (2008).

[12] N. P. Suh: *Axiomatic Design: Advances and Applications*, Oxford University Press, New York (2001).
(中尾政之, 飯野謙二, 畑村洋太郎共訳: 公理的設計, 森北出版 (2004)).

[13] N. P. Suh: *The Principles of Design*, Oxford University Press, New York (1990).

[14] 青島矢一, 武石彰: アーキテクチャという考え方, *ビジネスアーキテクチャ*, 第2章, 27-70, 有斐閣 (2001).

[15] 武石彰, 藤本隆宏, 具承桓: 自動車産業におけるモジュール化 (製品・生産・調達システムの複合ヒエラルキー), *ビジネスアーキテクチャ*, 第4章, 101-120, 有斐閣 (2001).

[16] *TOGAF*, https://www2.opengroup.org/ogsys/jsp/publications/PublicationsBySubjectType.jsp?limit=mainSubjectId-50:secSubjectId-50:statusId-1, accessed in 2011.

執筆者略歴

榮谷 昭宏（さかえだに あきひろ）

慶應義塾大学大学院システムデザイン・マネジメント研究科博士課程在籍 /NTT コムウェア株式会社勤務。1994 年学習院大学大学院修士課程修了後、NTT（日本電信電話株式会社）入社。約 15 年間情報システム開発に SE として従事。2010 年慶應義塾大学大学院システムデザイン・マネジメント研究科修士課程修了。この論文ではトレーサビリティ・マトリックスの考え方を提唱し、シナリオ構築・ソリューション構築を担当。

狼 嘉彰（おおかみ よしあき）

1968 年東京工業大学大学院理工学研究科博士課程修了。科学技術庁航空宇宙技術研究所、東京工業大学機械宇宙学科、慶應義塾大学理工学部システムデザイン工学科教授、慶應義塾大学大学院システムデザイン・マネジメント研究科委員長・教授を経て、慶應義塾大学システムデザイン・マネジメント研究所顧問。この間、米国 UCLA 客員研究員、宇宙開発事業団研究総監を兼任。専門は宇宙システムのダイナミクスと制御。日本機械学会フェロー。INCOSE Fellow。計測自動制御学会、日本航空宇宙学会、IEEE 等の会員。工学博士。この論文では数学的検証を担当。

神武 直彦（こうたけ なおひこ）

1998 年慶應義塾大学大学院理工学研究科修了。同年宇宙開発事業団入社。H-IIA ロケット搭載機器の研究開発に従事。欧州宇宙機関訪問研究員を経て、宇宙航空研究開発機構主任開発員として、宇宙機搭載ソフトウェアに対する独立検証および有効性確認に従事。現在、慶應義塾大学准教授、慶應義塾大学 VSE センター長。システムズエンジニアリング、プロセスアセスメント、宇宙システムおよびユビキタスシステムのデザインとマネジメントの研究等に従事。INCOSE、IEEE、情報処理学会等の会員。博士（政策・メディア）。この論文では研究統括を担当。

査読者との議論

議論1　構成学としてのこの論文

コメント（赤松 幹之：産業技術総合研究所ヒューマンライフテクノロジー研究部門）

この論文は、ソフトウエアの設計情報あるいは組織での情報がどのように伝播するかをトレース可能なモデル化手法の提案と理解しました。この手法によって、プロジェクトがどの程度複雑な

ものかを定量的に評価できることを示しています。

ソフトウエアを早く確実に構成するためには、複雑性を排除したプロジェクトを設計することが大事で、その意味において構成学に関係する論文です。しかし、情報の伝達効率を評価するためのトレーサビリティ・マトリックスの手法の説明が中心になっています。複雑性の比較ツールとしては活用できることは示されていますが、構成そのものの指針を得ることには直接的には活用できる例が示されていませんので、その点の加筆をお願いします。

コメント（中島 秀之：はこだて未来大学）

ここで提案されている手法が具体的に構成に役立つことを示してください。

回答（榮谷 昭宏）

「5.1.3 個々のマトリックスの相互依存性・難易度の低減と複雑性の評価」として記述しました。行列計算の特性から、トレーサビリティ・マトリックスの個々の行列はどのようにあるべきか説明し、具体的な事例を用いて相互依存性・難易度をどのように利用して最善の構成を作り上げていくか記述を追加しました。

議論2　この手法のマネージメントへの活用

コメント（赤松 幹之）

組織の複雑性が高いとマネージメントが困難になることは直感的には理解できますが、それはあくまで仮説であり、この手法によって、どのようにマネージメントが容易になるのかについて記載していただきたいと思います。具体的なマネージメントの仕方は構成学において重要なポイントであることから、この手法を使うことによってマネージメントがどのように容易になるのか、この手法で得られた指標を元にどのようにマネージメントをすれば良いのか等、具体的に示していただきたいと思います。

回答（榮谷 昭宏）

「5. トレーサビリティ・マトリックスを活用したプロジェクトマネージメントの PDCA サイクル」としてトレーサビリティ・マトリックスを用いたマネージメントの PDCA サイクルをシナリオとして記述しました。複雑性を指標としたプロジェクトの状況を評価する観点、また難易度や相互依存性に着目することによる個々の要素の状況を評価する観点について説明を追記しました。

議論3　PDCAサイクル

質問（赤松 幹之）

トレーサビリティ・マトリックスを使って PDCA サイクルを回すという利用方法が書かれていますが、サイクルを回しながらシステムマトリックス上の関係性や難易度を書き換えて複雑性を計算するのは、かなり面倒な作業のように想像します。ある領域の要素が別の領域のどの要素に使われるのかを把握したり、難易度の値の大きさをどのように決めるのか等、簡単ではないと思われます。現実的に PDCA を回すためには、その部分の工夫が必要ではないかと思いますので、その点についてのお考えを示していただきたいと思います。

回答（榮谷 昭宏）

ご指摘の点を追記しました。マネージメントの単位を検討し、適切な要素の粒度を選定すること、そしてその粒度にあった PDCA の周期を設定すること等を追記説明しました。また以下にシステムマトリックスの成分値の設定について、もう少し踏み込んだ見解を述べさせていただきます。

・難易度の設定について

難易度設定には、二つの難しさが考えられます。一つ目は定性的に要素の難易度を把握すること、二つ目はその定性的理解を定量化することとなります。

まず、定性的に難易度を把握することについて述べます。定性的に難易度を把握するにあたっては、プロジェクトの進捗状況やリスク事項等の情報を入力することになると考えております。既存のプロジェクトマネージメント技術としても進捗管理やリスク管理は実施されていることですので、難易度設定にあたっての入力情報としては問題なく整っていると考えております。実際、プロジェクトマネージャー、アーキテクト、または各担当のリーダークラスであるならば、感覚的には各要素の難易度の変化を把握していると考えております。例えば、多くのプロジェクトでは、誰がキーマンであるとか、どのツールに問題があるとか、どの作業がキーポイントになる等、プロジェクト内で交わされる会話の節々でプロジェクト構成要素の難易度に関する議論は多くなされています。したがって、定性的に難易度を把握することに特に大きな障害はないと考えております。

しかし、そのリスク事項を難易度に定量化することには課題があると考えております。この論文で提示した指針で、誰もが同じような評価をできるかまだ検証が不十分であり、その改善は今後の課題と認識しております。

・相互依存性の設定について

作業標準のようなものが存在しないプロジェクトにおいては、相互依存性を整理することもとても困難な作業となると予想されます。そのように作業標準が存在しないプロジェクトでは、そもそも既存のプロセスを中心としたマネージメントも困難なはずです。そのような意味で、今回提案させていただいたモデルを用いることは、当初不慣れさから生じる面倒さを感じるかもしれませんが、プロジェクトの状況を把握するツールとして有効に機能すると考えます。また作業標準をすでにもっているプロジェクトでは、一般的に作業標準では作業と成果物、また各担当の役割を定義しているものであるので、その段階から相互依存性を整理することは可能であり、その整理が一度できてしまえば、それをテンプレートとして、個々のプロジェクトでカスタマイズし、使い回すことに大きな障害はないと考えております。

・全体をとおして

ご指摘のように少なくとも当初はとても難しいマネージメント技術に感じられることもあるかもしれません。今までのプロジェクトマネージメントでは、プロセスを中心とした進捗状況の管理、およびそれに則した予算の管理が主体であったため、その観点から生じている問題を分析的に調べ、原因の対処を行ってきました。しかし、この手法では、モデルをもとにして個々の要素の難易度と相互依存性によってプロジェクトで生じている問題を構成的に理解し、全体を俯瞰した視点から対処を行っていくという考え方となります。つまり、今までのプロジェクトマネージメントは、分析的であるが故に個々の要素を中心とした個別最適なソリューションに傾倒してしまいがちであるのに対して、ここで提案しているものはプロジェクトの全体最適となるソリューションを思考するツールになると考えております。そういった面で、思考方法の違いが障害となってしまうケースは有り得るかもしれませんが、思考の切り替えさえできれば、大きな問題はないように考えます。難易度のところで申し上げたとおりプロジェクトマネージャー等は感覚的に把握していることですので、例えば EVM（Earned Value Management）の実行よりも容易ではないかと考えております。

議論4　終了したプロジェクト

質問（中島 秀之）

「完全に情報移転が終わった要素間ではその相互依存性が無くなっていく」とありますが、終了したプロジェクト同士には情報依存性が無いと考えるべきなのでしょうか？たとえばある新しい技術が発見されて、プロジェクト１の成果を改良した場合に同じ

ことがプロジェクト２の成果にも適用できるということはないでしょうか。

回答（榮谷 昭宏）

どのような単位でプロジェクトを定義するかによって、相互依存性の有無に対する考え方は変わってきます。

新しい技術を確立したプロジェクト１と、次のプロジェクト２を合わせて一つのプロジェクトとしてとらえた場合は、相互依存性はなくなるとは考えません。しかし、プロジェクト１とプロジェクト２を別々のプロジェクトとしてとらえた場合は、プロジェクト１において相互依存性は無くなり、プロジェクト２では例えば担当者の習熟度向上により、担当者の難易度をプロジェクト１よりも低く設定する等により、その情報移転を評価し反映します。