慶應義塾大学学術情報リポジトリ
Keio Associated Repository of Academic resouces

Title	行列を用いた構造分析による高品質・高生産性を実現するプロジェクト設計：情報システム開発の10年後を見据えて
Sub Title	Project design structure for high quality and high productivity by matrix : scenario for the next 10 years in software development projects
Author	榮谷, 昭宏(Sakaedani, Akihiro) 手嶋, 龍一(Teshima, Ryuichi)
Publisher	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
Publication year	2009
Jtitle	
JaLC DOI	
Abstract	
Notes	修士学位論文. 2009年度システムデザイン・マネジメント学 第14号
Genre	Thesis or Dissertation
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO40002001-00002009-0017

慶應義塾大学学術情報リポジトリ(KOARA)に掲載されているコンテンツの著作権は、それぞれの著作者、学会または出版社/発行者に帰属し、その権利は著作権法によって保護されています。引用にあたっては、著作権法を遵守してご利用ください。

The copyrights of content available on the KeiO Associated Repository of Academic resources (KOARA) belong to the respective authors, academic societies, or publishers/issuers, and these rights are protected by the Japanese Copyright Act. When quoting the content, please follow the Japanese copyright act.

修士論文　　2009 年度

行列を用いた構造分析による高品質・高生産性を実現するプロジェクト設計

－情報システム開発の 10 年後を見据えて－

榮谷 昭宏

（学籍番号：80833204）

指導教員　教授　手嶋 龍一

2010 年 3 月

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

システムデザイン・マネジメント専攻

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　個人研究論文

論　　文　　要　　旨

学籍番号	80833204	氏　名	榮谷 昭宏

論 文 題 目：

行列を用いた構造分析による高品質・高生産性を実現するプロジェクト設計
－情報システム開発の 10 年後を見据えて－

（内容の要旨）

1. 研究の背景

近年、情報システムは社会インフラとしてなくてはならないものとなっている。しかし、その品質は脆くそのトラブル事例は最近の新聞報道でも多くみられる。多くの技術者・研究者がソフトウェアの品質・生産性向上に向けた努力を行っているが、未だにその解決の兆しは見えない。10 数年ソフトウェア開発業務に従事してもその進歩を感じることは出来ず、今まで通りのやり方で進歩は望めるのであろうか。

2. 情報システム開発の現状

多くのステークホルダー、多くのテクノロジー、多くの開発者が関連して開発を行うことで、プロジェクトがひとつの複雑なシステムになっている。

そのため、必要な設計情報が様々な要因で欠落または湾曲し、必要なところに必要な情報が正確に届かなくなっている。結果として設計変更や仕様変更も多発し、益々プロジェクト全体のコントロールを難しいものにしている。

3. 解決すべき課題

多くの要素およびその関係性を見える化し、その情報伝達ルートをコントロールする手法が必要である。

4. ソリューション

プロジェクトの構成要素として Needs – feature – requirement – function – component – artifact – activity – team を抽出し、この要素間の関係性をネットワーク図により視覚化し、その関係性を行列化することで定量評価を可能にする。それにより、どのように改善を図るべきか明確にする。

5. 情報システム開発の 10 年後と対策

将来的にも情報システム開発の構成要素が益々複雑化してくることが予測される。その複雑化するプロジェクトを操作可能にするには、要素間の関係性を極力対角化に近い関係性にすることが必要である。構成する要素間の関係が全行列で表される関係となったとしても、その箇所は局所化する必要がある。

6. ロバストなプロジェクト設計

情報システム開発のやり方を自動車開発に学ぶ。自動車開発では自動車を構成する要素毎に組織を分け、要素内に生じる複雑な関係性は要素内に閉じたものとしてクラスター化している。その要素間の複雑な関係全体はチーフエンジニアがコントロールすることで、複雑な全体の調和を保っている。

情報システム開発でも、要素技術の複雑性のコントロールと全体調和のコントロールを別々に分担する手法は学ぶべきであり、将来に向けた必須事項である。

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　個人研究論文

SUMMARY OF MASTER'S DISSERTATION

Student Identification Number	80833204	Name	AKIHIRO SAKAEDANI

Title

Project Design Structure for High Quality and High Productivity by Matrix

－ Scenario for the Next 10 Years in Software Development Projects－

Abstract

1. Background of research

In recent years, the information system is indispensable as the society's infrastructure. However, the quality is fragile and a lot of recent press reports are seen as for the trouble case. Although the sign of solution has not yet been shown, a number of engineers and researcher are making efforts for the quality and the productivity enhancement of software. I still cannot see the advancement even though I have been engaged in the software development business for ten-odd years. If the traditional system stays the same, advancement that exceeds the current state cannot be hoped for.

2. Current state of software development

In the development project, many stake-holders, technologies and developers participate and make the project as one complex system. Therefore the design information can be missed by various factors, and thus the necessary design information can not be reached to necessary point accurately.

As a result, the design and the specification frequently happen to change which makes it more difficult to control the whole project.

3. Problem

A number of elements and the relations are not yet visualized. And the approach for controlling the information transmission route is missing.

4. Solution

As the components of the project, the following must be extracted: needs, feature, requirement, function, component, artifact, activity, and team. By visualizing the relations between these elements by using the network figure, and making the relation into matrix, the quantitative evaluation is enabled. As a result, it clarifies how to improve the system.

5. After ten years of software development and action plan

It is forecasted that the components of the software development project become complex more and more in the future. In order to make the complex project operable, "diagonal matrix" must be applied to components. Even if the relations between composing elements are shown by "general matrix", it is necessary to localize each part.

6. Design of robust project

The way of software development project can learn from the automotive development project. In the automotive development project, the organization is divided into elements that compose the architecture of car. Teams by clustered architecture control the complexity of each element. And the chief engineer controls the entire complex relations between the elements thus the entire complex system stays in harmony.

In the software development, it is necessary to learn the technique for controlling the complexity of each element and the entire harmony, and it is indispensable for development in the future.

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　個人研究論文

行列を用いた構造分析による高品質・高生産性を実現するプロジェクト設計
― 情報システム開発の 10 年後を見据えて ―

Project Design Structure for High Quality and High Productivity by Matrix

― Scenario for the Next 10 Years in Software Development Projects―

榮谷　昭宏
Akihiro Sakaedani, Keio University Graduate School System Design and Management

In the modern society, the information system is an extremely important part of the social infrastructure. It is therefore necessary for us to constantly improve the quality and reliability of information system. At the present, research tends to focus narrowly on technological aspects such as software engineering. Considering the tremendous variety of products, enterprises, and people in the information service industry, it seems that we need to take a new approach to this problem that analyzes the information service industry for complex system.

In this research, it focused on the relations between elements that composed the software development project. And it revealed to the ideal type for the high quality and the high productivity. Finally, this paper will try to clarify what systems integrators should be doing now in order to realize this future vision.

***Key Words*:** Information service industry, Axiomatic design, QCD (Quality, Cost, Delivery), Software development, Matrix, Scenario planning

1.　はじめに

近年、東証や全日空、JAL、JR、そして銀行 ATM といった社会インフラともいえるシステムのトラブルが大きく報道されるようになった。そのような報道の背景には情報システムのトラブルが 10 年前の位置づけとは異なり、情報システムを構成する“ソフトウェア”が社会生活を構成するインフラを担い、その品質が非常に重要なものになってきたことがあるのではないか。将来的にも情報システムへの依存度が高まることはほぼ間違いのない事実であろう。しかも更に複雑に情報システム間が連携し、そして、更に大規模な情報システムが社会基盤として我々の日常生活を支えるようになるであろう。しかし、そのように考えたときに今の情報システム開発のやり方で良いのであろうか、また、今のような情報システムの品質で良いのであろうか。10 年後を見据えたときにどのような将来が待ち受けているのだろうか。詳細は後述するが将来的に以下の２つは確実に実現していると言える事象であると考える。

- 情報技術（IT）は 2020 年になっても存在し、発展している。
- 情報技術（IT）は今後さらに社会インフラとして重要度を増してくる。

しかし、逆に将来的に最も不確かで且つ情報システム開発産業に影響を与えると考える事象は

- ソフトウェア開発費用の削減、品質向上に向けた技術（生産技術）はどのように発達するのか。
- 社会インフラ化した情報システムは、それに準ずる品質を満たすことができるのか。

という２つであると考えた。以上のことから、下図に示すように４つのシナリオが想定される。

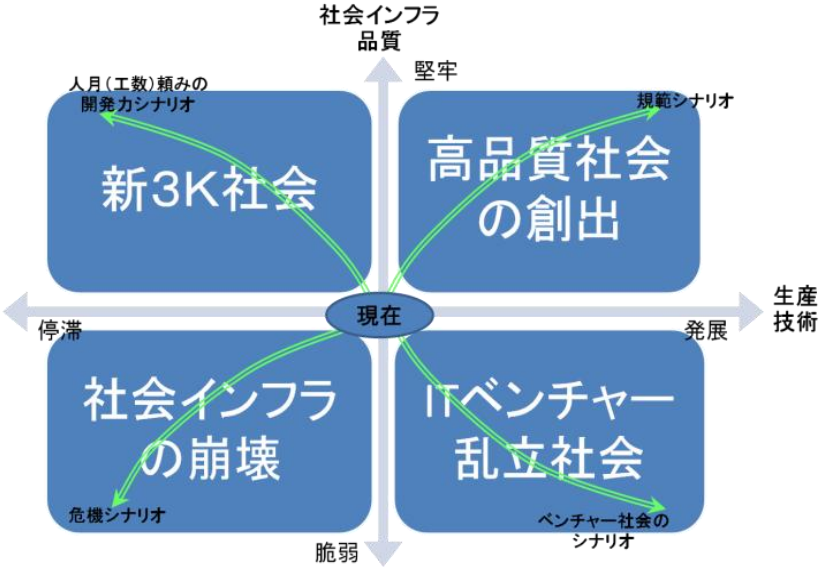


図 1　情報サービス産業の将来（2020 年に向けたシナリオプランニング）

（角和昌浩（2005）を参考に作成）

規範シナリオでは、高品質社会の創出という象限名のとおり、一番理想的な将来像を想定している。新３Ｋ社会と名付けた象限では生産技術は現在以上に発展することもなく停滞気味であるが、それを補完するために工数に依存した生産性・品質の向上を達成しようとしている社会を想定している。そして IT ベンチャー乱立社会という象限では、生産技術が特に標準化されることもないまま色々な技術が IT ベンチャーにより生み出されて生産技術は発展傾向にあるが、それを永続的に維持管理していくだけの体力がベンチャー企業にはなく、結果として社会インフラとしての品質を脆弱なものにしてしまうような社会を想定した。最後の象限として社会インフラの崩壊として、危機シナリオを考えた。このシナリオでは生産技術が停滞し、社会インフラの品質も脆弱化していってしまうことを想定している。

本研究では、このまま成り行き任せの状況では“危機シナリオ”を辿ってしまうリスクもあるところで、如何に規範シナリオを描いて発展していけるのか考えていくことをテーマとする。

目次

1. はじめに 4

2. 本研究の背景 8

3. 先行研究の状況 12

4. 分析フレームワークなどについて説明 13

5. 日本の情報サービス産業の現状について 14

6. 日本の IT 産業に見る情報のカオス化による影響分析 25

◆ 各業務プロセスの状況について 25

7. 問題点のまとめ 30

8. 情報の滞留原因 31

◆ そもそも情報はどのように伝わるのか 31

◆ これまでの議論の整理とソリューション検討に向けて 31

◆ ソリューション・フレームワーク　～メソドロジー～ 32

① プロダクト品質から組織活動品質へ（旧来のシステムエンジニアリングと比較して） 37

② 本研究で考えたモデルの特徴 37

③ コンセプト説明（情報を円滑に伝える仕組みとは） 39

9. ケーススタディー 55

◆ ユースケース 56

◆ 企画フェーズ（ネットワーク図） 58

◆ 企画フェーズ（マトリックス） 59

◆ 開発フェーズースクラッチ開発（ネットワーク図） 62

◆ 開発フェーズースクラッチ開発（マトリックス） 63

◆ 開発フェーズーＰＫＧ利用 65

◆ 開発フェーズーＰＫＧ利用（マトリックス） 66

◆ 開発フェーズージェネレータ（ネットワーク図） 68

◆ 開発フェーズージェネレータ（マトリックス） 69
◆ 比較分析（コスト、生産性） 71
◆ 比較分析（品質） 72
◆ 比較分析（開発期間） 75
◆ ケーススタディーまとめ（情報伝達の円滑度） 78
10. 将来に向けたソリューション検討 79
◆ 外部環境の整理とソリューション検討 79
① シナリオ作成方法 79
② SEPT 分析 79
③ シナリオ・プランニング 82
◆ 内部環境の整理とソリューション検討 91
① 現状の課題と他業種の状況 91
② 情報システム開発プロジェクトの留意点と組織の定量的議論 95
◆ ソリューション　～IT 業界ではどうすべきか 98
① 外的環境分析と内的環境分析から 98
② To-Be モデル 99
11. まとめ 103
12. 今後に向けて 104
13. 謝辞 107
引用文献　108

2.　本研究の背景

第 15 回企業 IT 動向調査 2009 によると、システム開発プロジェクトの工期・予算・品質に対して、多くの問題が残存していることが分かる。

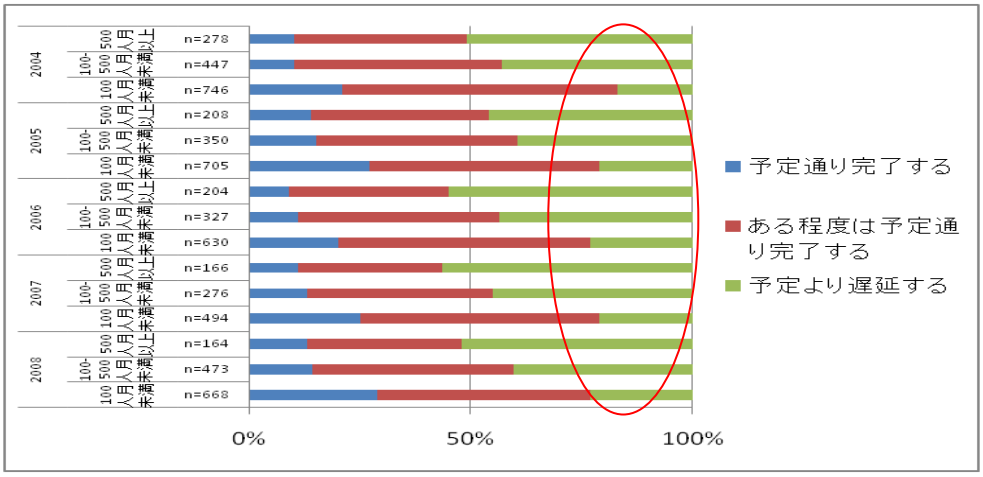


図 2　システム開発の工期　　第 15 回企業 IT 動向調査 2009 より

システム開発の工期について、500 人月以上の大規模系のプロジェクトでは予定通り完了するのは大規模プロジェクト中の 13%に過ぎない。残りの 90%近くが遅延する見込みであり、特に明確に予定より遅延するというプロジェクトは 52%に達する。

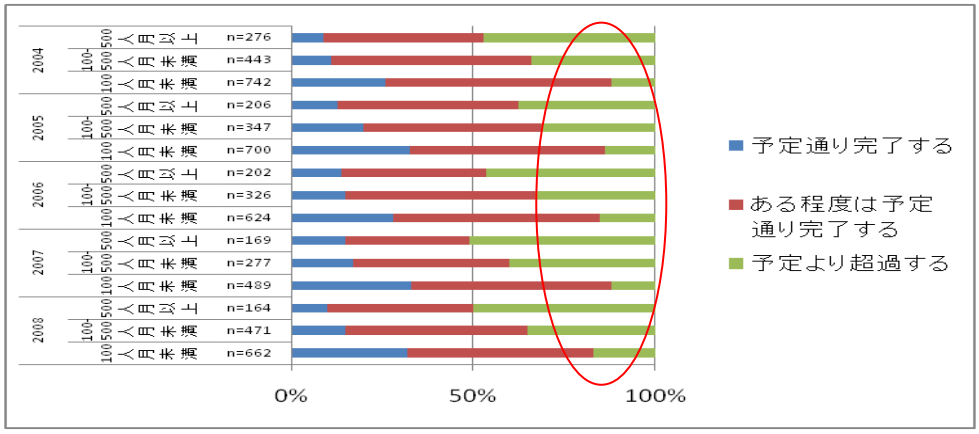


図 3　システム開発の予算　　第 15 回企業 IT 動向調査 2009 より

しかも、開発予算についても予定より超過するプロジェクトが 50%近い状況である。この様な実情では、何とかプロジェクトをなんとか完成させて稼働させることが精一杯で、十分な品質を担保するに至っていないことを想像することは非常に容易なことであろう。

事実、品質に対する不満も多い。“ある程度満足している“という回答をどのように理解すべきか議論の余地はあるが、少なくとも“満足している“という比率は 10%前後に留まっており、そのレベルの低さは、改善への動機付けを得るには十分な調査結果であるように考える。

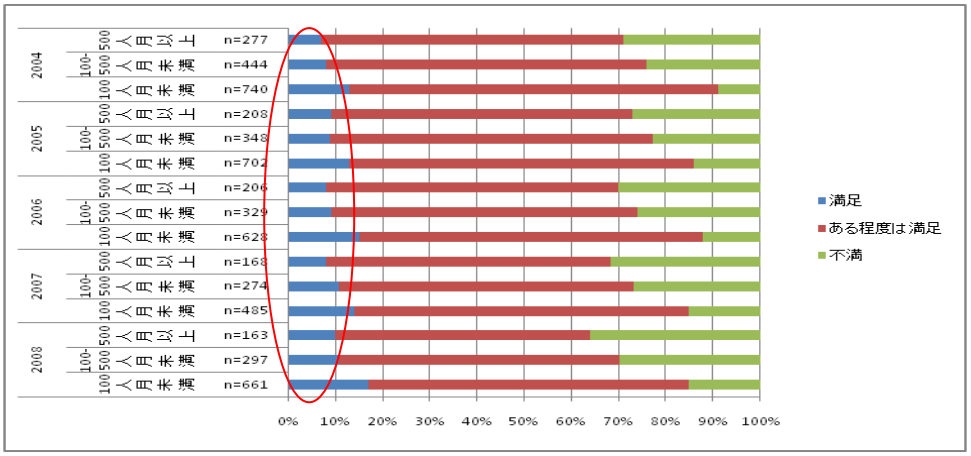


図 4　システム開発の品質

そこでまず、ソフトウェア品質の不安定さは何故起きているのか、おそらくそれは自動車・建設業界などのような成熟した生産技術を確立できていないことに原因があるのではないかと考えた。では、何故ソフトウェア業界は生産技術が未成熟なのか。まず他業界と異なり、製品そのものが視覚的に見える物ではないという特徴がある。そのため、“設計思想“を文書によって各作業者へ正確に伝えていくことが非常に重要なこととなっている。しかし建築家もデッサンやサンプルモデルにより視覚的にイメージを伝えているように、現実的には文書だけで“設計思想”を伝えることは非常に困難である。特に数百名規模で開発を行うような場合は困難極まりない作業となることは容易に想像できるであろう。ソフトウェア開発においても UML などを用いて極力視覚化する努力はしているが、図示するだけで十分に意図が伝わるまでには至っていない。

上記特徴により、他の業界と比較してまず初期段階での難易度が高くなっているが、更にソフトウェア開発を構成する“システム”の各要素の複雑性、関連性が問題をさらに難しくしている。

開発作業（設計試験など）の制約条件は大きく捉えて、利用技術、要求仕様、開発費用、開発期間、開発組織（人）、そして開発方法の 6 つあると考える。しかし、ソフトウェアの基本的な特徴及び上記の 6 要素、その全てを上手く調和させ、且つ個々の要素毎に質の向上を図っていくことで高品質のソフトウェアを開発することが可能となるが、そこに到達するための障壁が高い。

例えば“利用技術”は他の業界と比較してもその進歩は非常に早く、追従するだけでも大変な労力を要する。反面、習得した技術が数年で廃れたり、それが革新的技術であるほど逆に品質が悪かったり、ということも多い。また利用技術を目利きするためにも、新技術習得のための“開発組織（人）”への教育は怠るわけにはいかないが、そのために十分な期間・費用をかけることはできない。などというように個々の要因が複雑にしかも相反する関係を持って絡み合っている。そのため、個々の開発案件の事情により各要素の複雑性・関係性も随時変化し、多くの場合安定した製品開発の妨げとなっている。つまり、現実の開発では作業の“定型化”、“再帰化”が困難なのである。

過去 15 年を振り返ってみても、開発の質はどれだけ向上したであろうか。1989 年に冷戦が終結し、米国の国防関連技術のひとつであったインターネット技術が商用ベースにも利用されるようになり、日本で一般的に普及し出したのが 1995 年前後である。そのころから考えると、情報技術（IT）を利用する立場からは非常にその利便性が高まっていると言えるが、情報技術（IT）を用いたサービスを開発する側はあまり変化していないと考える。1995 年前後では日本の大手システムインテグレーターでさえ、その設計作業はまだワープロな

どを利用しており、設計情報の共有は基本的には“紙“によってなされることがほとんどであった。s コーディングレベルの技術では COBOL から Java など技術に変遷はあっても、実質的な作業内容は相変わらず人海戦術のケースがほとんどである。10 年以上前からソースコードを自動生成する仕組みの研究は行われ、また商品として提供されている技術もあるが、その技術がクリティカルなシステム開発に用いられるレベルまで信頼できるものにはなっておらず、また一般的にみても開発技術者に普及するまでに至っていない。これからの 10 年、情報システムを作る側がこのような状況から進歩しない限り、日本の労働人口が減少し人海戦術だけで何とか動かしているような情報システムでは社会基盤となるクリティカルなソフトウェアは構築できなくなってくるであろう。

ある限られた業界の開発においては作業の定型化が図られ、開発作業を行う毎にその質が向上していくような仕組み化が機能しているところはあるが、たいていの業界ではその仕組み化が上手くできていない。そのため何度も同じようなトラブルに遭遇し、何度も同じように結局は人手をかけて徹夜作業により乗り切ってしまうことで対応を済ましてしまい、開発組織・ソフトウェア開発業界として進歩がないのが現状である。

また別の観点ではあるが、次のことは IT 産業発展のための考慮も必要である。例えば社会基盤となる技術でなく、最新技術を駆使して構築するシステムも存在する。それはベンチャー企業が自分たちの開発した技術を社会に提供し、世の中を変えていこうというチャレンジの場であり、そのような情報システム、情報技術（IT）にまで初めから高い品質を求めることは無理難題であると考える。そのような技術でいきなり社会基盤を支えるようなシステムに導入するのではなく、そういったシステムの周辺などで、技術品質の安定性を確保して技術者の中で製品価値を確立することが必要であり、そのことが小さなベンチャー企業を大きく育てるステップとなると考える。

あくまでもそのように技術の適用領域を考慮することが重要であり、また社会基盤を担うシステムを開発する場合の導入技術の見極めをすることが重要であると考える。新しい技術を利用するターゲットを絞ることなく、全般的にその技術を投入することは、せっかく創出された優良な技術をも花開くことなく時間とともに消滅させてしまうことに成りかねない。

もっと真剣に業界全体として定型化及び再帰性のあるシステムとして生産技術を構築し、質の向上を図るべきである。その結果、社会基盤と成りうる品質のソフトウェアが作られ、社会生活の向上に寄与することができるであろう。

現代社会において情報システムは非常にクリティカルな社会基盤となっている“社会中枢システム”とも呼ぶべきものであるにも関わらず、このような実情ではこれから 10 年後さらに情報技術（IT）に依存度が高まる社会が予想されるが、社会基盤として成立するだけの品質を保持できているか非常に不安である。

過去に情報システムの信頼性・品質の向上のためにソフトウェア工学においても日々研究が進められているが十分な解決策を得られていない。今後も様々な製品や企業、またはそこで働く様々な人々がいる複雑系の中では、単純に個々の要素技術を中心した工学的なアプローチだけでは十分な改善は望めず、他の視点も合わせて工学的なソリューションを活用することで有益に機能していくと考えられる。

そこで本論文では情報サービス産業をひとつの社会システムと捉えて、システムエンジニアリング的分析設計を行うこととした。

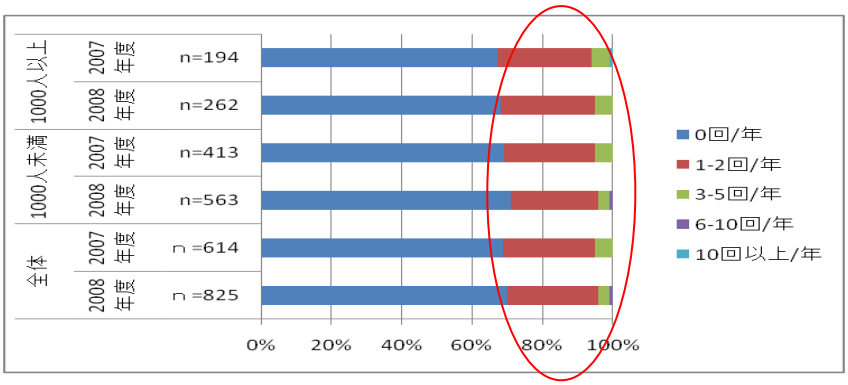


図 5　システム障害による事業中断回数　　第 15 回企業 IT 動向調査 2009 より

現実的に上図のとおりシステム障害による事業中断回数 0 回／年の回答比率は 70%程度でしかない。社会インフラ化した現在、30%近くの企業で情報システムの障害停止が年に 1-5 回も発生していることはシステムの脆弱性を示すものであり、現代社会が如何に軟弱で“ソフト”な情報システムの上で成り立っているのか、ということを認識すべきである。もしシステムが停止するとしてもそれは障害による停止ではなく計画停止でなければならない。障害による突然の停止の不愉快さは毎朝の電車の遅延以上なのは想像するのも容易であろう。

3.　先行研究の状況

経産省では当然政策立案の立場から、現状の情報サービス産業についてのレポートを発表している。しかし、その内容はあくまでも情報技術（IT）を利用する一般の人々からの視点で分析されており、技術を利用することによってどれだけ社会生活が変革するか、利便性が向上するか、という観点が主な内容である。この傾向は総務省や文科省の情報技術（IT）に対するレポートも同様であり、情報技術（IT）の質や生産性を高めるために、または IT 業界内部の改革について言及したものは経産省関連の IPA（独立行政法人 情報処理推進機構）から出されているレポート程度に限られる。その IPA のレポートについてはソフトウェア工学に近い議論が多く、開発には複数の企業が連携していることなど現状の複雑な要素を考慮した議論は少ない。

但し、ようやく 2009.5.28 に経産省から「高度情報化社会における情報システム・ソフトウェアの信頼性及びセキュリティに関する研究会の中間報告書」が発表され、この中間報告書ではソフトウェア品質を保証するための仕組みや、ソフトウェア工学の発達にも言及し、そして契約形態に対する問題についても指摘している。それまでの情報技術（IT）を利用する立場から、情報技術（IT）を作り出す立場の視点で報告書がまとめられたことは意義のあることと考える。しかしながら、その内容は IT 産業を取り巻く外部環境の変化を考慮しておらず、数年先にまで有効な提言になっているか検証が必要である。また、情報システムのアーキテクチャに対する問題、開発体制における専門家育成の問題、などに対する問題にまで踏み込んでおらず、今後さらに検討を進める必要がある。

また、その他の多くの IT 産業に関するレポートは、産業全体を俯瞰したものではなく、産業を構成するある企業に特化した分析が多い。例えば過去にはマイクロソフト社に関する書籍は多かったが現在では Google や Amazon に関する書籍などを良く見かける。ある企業に特化した書籍はそこでの開発状況を紹介していたりするが、そこにある背景などまでを深く分析し日本企業に展開するところまでには至っていない。

その他の観点では、ある特定の要素技術に着目したレポート・書籍が多いが、複数企業にまたがって業務プロセスを分担して全体としてはひとつの情報システムにまとめ上げているという開発の実情を踏まえていない。そのため、実際に“効く”処方箋を提示したレポート・書籍はほとんど存在しない状態である。

繰り返すが、現代の大規模情報システムでは、何社もの関係企業が協力してひとつの情報システムを組み上げている。それを踏まえて、今までのようにそれを単にひとつの“プロジェクト“としてのみ分析し論じるのではなく、そのプロジェクト自体をひとつの社会システムと診ることによって、全体を俯瞰した上で問題点を洗い出し、その解決策を検討することで、現実に応用可能なソリューションを提示することを本論文の狙いとする。

4.　分析フレームワークなどについて説明

社会システム全般を分析する独自フレームワークを提案する。静的ビュー（時間軸を現在に固定して考えた観点）と動的ビュー（時間軸を動かしながら考えた観点）の２つの分析フレームワークを用意した。

静的ビュー（図１）では、社会システムは必ず何かしらの生産活動を行っており、その活動を制御・管理する機能①と活動の実行ステップ②、そして実際にそのプロセスに基づいて実行する機能及び実行結果③の３つで成り立っているという観点から分析を行う。

それぞれを①プロセス管理層、②プロセス層、③プロセス実行層と名づけ、３層に分離して、個々の層内及び層間の関係性を分析し、課題を抽出する。また抽出した課題の良否を判断するために、価値観的側面を考慮するようなフレームワークとした。

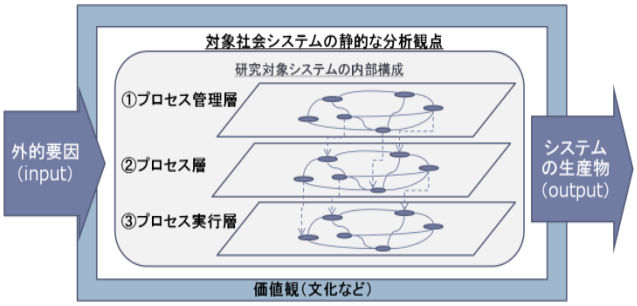


図 6　静的ビュー_分析フレームワーク

動的ビュー（図 2）では、研究対象とするシステムのライフサイクルに着目して分析を行う。例えば、システム誕生当初とシステムの衰退期では、必然的におきる問題もあれば、次のフェーズに移行する上で必ず対処しなければならない課題も異なる。よって、対象とするシステムのフェーズが何処なのか考慮せずに静的視点で取り上げた課題のみを論じることは、採るべき対策の方向性を誤るリスクを増大させることにもなる。そのようなことを防止するために動的観点の分析を取り入れた。小規模なシステムと大規模なシステムそれぞれについて、①立上げ期、②成長期、③安定期、④衰退期のフェーズ毎の特徴を整理した。

			システムのライフサイクル			
			①立上げ期	②成長期	③安定期	④衰退期
システム規模	小規模	プロセス管理層	役割分担明確	成熟	小規模化	小規模化
		プロセス層	混沌	ある程度の秩序化	一部のみ機能	一部のみ機能
		プロセス実行層	漠然	具体的	品質向上	現状維持or劣化
	大規模	プロセス管理層	実質的リーダの出現	役割分担の明確化	成熟	小規模化or混沌
		プロセス層	混沌	秩序化	全体効率的	一部のみ機能
		プロセス実行層	漠然	具体的	品質向上	現状維持or劣化

図 7　動的ビュー_分析フレームワーク

5.　日本の情報サービス産業の現状について

本章では産業構造を静的・動的に分析し、その課題を抽出していくが、一口に情報サービス産業といっても様々なカテゴリがある。はじめに、本研究の対象は総務省統計局の日本標準産業分類による情報サービス産業（中分類）のソフトウェア業（小分類）に位置付けられる受託開発ソフトウェア業（細分類）である。更に受託開発ソフトウェア業は、“情報システム開発（エンタープライズ系）”と“組込み開発”の２つのカテゴリに分類することが可能である。本研究では“情報システム開発”を中心とした議論を行う。参考までに、２つのカテゴリの違いについて説明する。“組込み開発“とは、携帯電話や iPod など、あるハードウェアに内蔵されてそのハードウェアの制御を目的とした特定の機能（ソフトウェア）を開発する場合を指している。情報システム開発とは、主に企業内の事務処理を自動化する場合が多く、サーバーなどを用いて事務処理データを流通、加工、保管などを行うシステムの開発を指している。

総務省の定義によると、受託開発ソフトウェア業は「顧客の委託により、電子計算機のプログラムの作成及びその作成に関して、調査、分析、助言などを行う事業所をいう。」とあり、受託開発ソフトウェア業にはプログラム作成業、情報システム開発業、ソフトウェア作成コンサルタント業が含まれるが、パッケージソフトウェア業やゲーム用ソフトウェア作成業などは含まれていない。

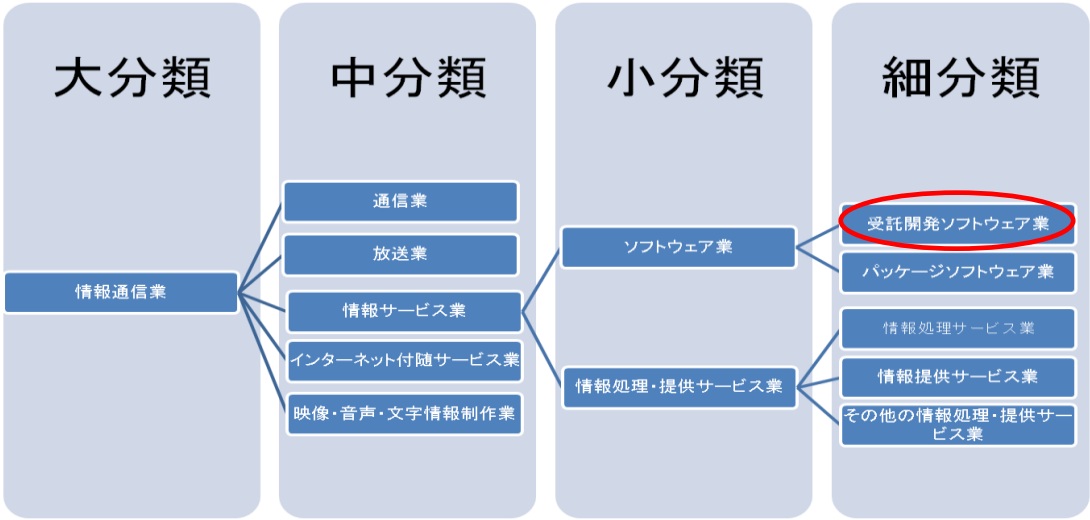


図 8　情報サービス産業の定義と研究の対象範囲

総務省　統計局　日本標準産業分類より

http://www.stat.go.jp/index/seido/sangyo/index.htm

ここからは更に受託開発ソフトウェア業の実情について静的なビューを設定して、つまり今現在の産業構造に着目して IT 産業の分析を行っていく。分析を行うにあたって、静的分析フレームワークの考え方を用いて、作業担当・作業プロセス・成果物の３層に分離してその関係を整理する。

Ｖ字モデルの流れで、企画（業務改善分析）→要求管理→分析→設計→製造→単体テスト→結合テスト→総合テスト→受入テストの作業プロセスで作業は進められ、各プロセスに対応して、それぞれ次の成果物を作成する。企画書→要求仕様書→分析モデル→設計モデル→実装→単体テスト結果→結合テスト結果→総合テスト結果→受入テスト結果、といった成果物を生成する。

そのときに企画プロセスは顧客、要求管理はシステムインテグレーター、分析は１次請負企業、設計及び製

造は２次請負企業、単体テスト結果と結合テスト結果は同様に２次請企業、総合テスト結果は１次請企業、そして受入テスト結果は顧客によって作成される。

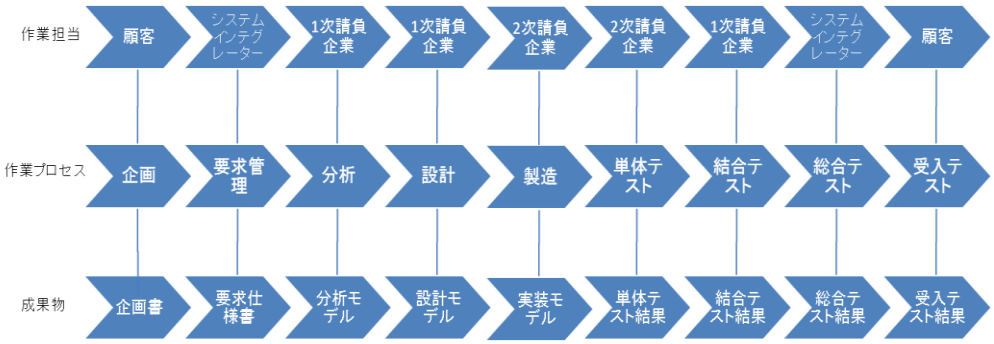


図 9　各開発工程における作業担当-作業プロセス-成果物の関係

しかし、現実は企画プロセスをシステムインテグレーターが大半の作業をサポートし、要求管理プロセスは１次請負企業が大半の作業をサポートする、同様に分析・設計などもすべてひとつずつ繰り上がった形で作業が進められる。

2004 年情報産業白書「3. 変革要因による収益構造変化への対応」で述べられているとおり、「これまで情報サービス業者の主な収益源は、開発工程であり、次いで運用工程であった。しかし顧客企業の業界への新規参入や競争環境のグローバル化などにより、開発工程での収益性が低下する傾向にある。
情報サービス業者は将来の収益源を、「コンサルティングなどの上流工程」「運用・サービスなどの下流工程」に求める傾向が強まる。

同白書の中では、下請型企業の生き残り戦略…①自律型への転換戦略、②製造特化による差別化集中戦略、元請型の競争戦略…①元請型特化戦略、②自立型への転換戦略」と紹介している。

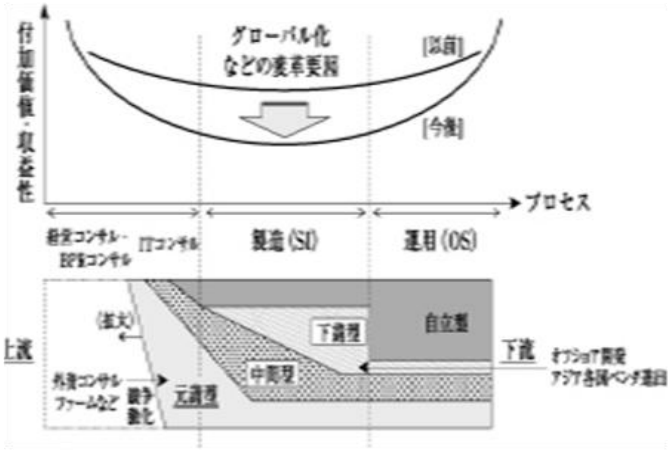


図 10　企業類型を考慮した提供サービスと収益性の関係

出典　情報サービス産業白書　2004 年版より

このような傾向を示す原因は以下のように考えることができる。企業内組織であった情報システム部門を分社化し、その経営資源から他社案件の受注に結びつけることで経営の多角化を図ろうとする風潮が強い時期があった。その時に本来ならば企業内の業務プロセス分析を行い経営状況に応じた IT 戦略を検討するべき組織

が、分社化することにより企業内に存在しなくなってしまったようなところもある。10 年も経ってしまうとそのような分社化した企業にも本来熟知しているべき親企業の業務内容も十分に把握しているとは言えない状況になってしまっている。結果として、親企業と共同でその旧情報システム部門が現状の問題を企画書としてまとめ、情報システムの開発に着手していく。というような流れになっている。下図のとおり、今後もその傾向は続くであろう。各企業ともに情報システム子会社やその他企業へのアウトソースを増加させる傾向にあることは明白であり、情報技術（IT）の重要性は多くの企業で認識されているが、それを抱えることはしない方向である。自社内で対処することのほうがシステム開発や運用という面からは良い面も多いと言えるが、その他企業の財務会計、マーケティングなど様々な観点で考えると、自社開発がベストなソリューションとは言えない面もある。開発量がコンスタントに一定量を保つことができないならば技術者を常時自前で抱えることは企業として非常に負担の大きいことである。そのように今後も子会社やアウトソースが継続し発展していくことを前提にして、どのように高品質な情報システムを構築すべきか検討することが必要であると言える。

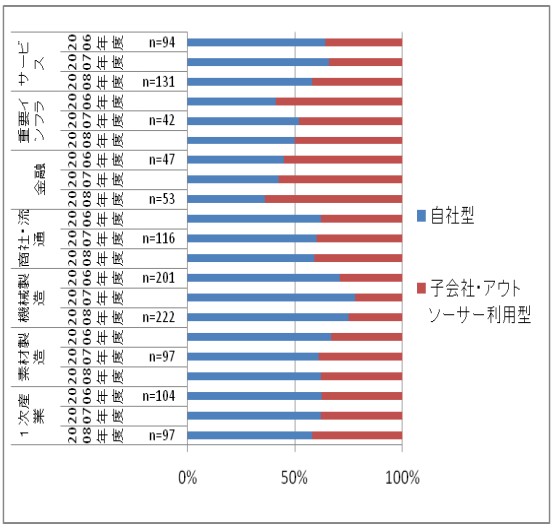


図 12　業種グループ別情報子会社・アウトソーサー利用状況

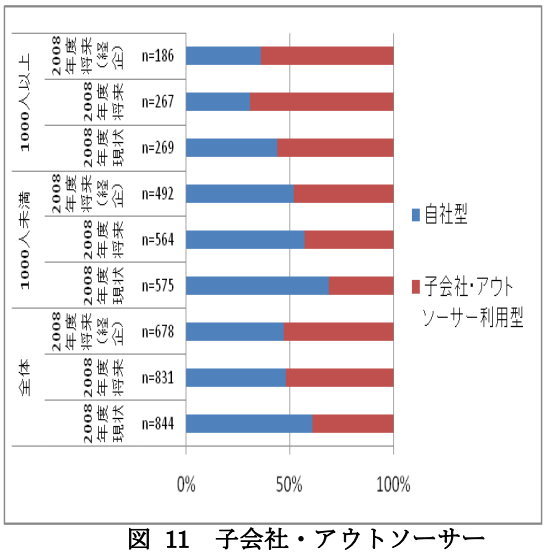


図 11　子会社・アウトソーサー利用の方向性（現状と将来）

第 15 回企業 IT 動向調査 2009 より編集

情報子会社やアウトソーサーとの関係の実例を紹介する。契約上プライムコンストラクターはシステムインテグレーターであるが、その作業のほとんどは顧客からの発注内容の整理（どのような問題が業務上にあり、何を改善する必要があるのか整理すること）に多く割かれており、本来システムインテグレーターとして果たすべき要求管理を 1 次請負企業に依存してしまっているというのが現状である。その結果、作るシステムの要求仕様や分析・設計仕様は下位の請負企業に多くを依存する。つまり、成果物は契約上システムインテグレーターが作成したことになるが、実質的な作業プロセスとしては委任先企業に行ってもらい、出来た成果物をレビューし、チェックすることで作業プロセスを完了している。

また、実作業担当がずれているため当然のことではあるが、成果物自体の内容も本来上位の工程でその設計書上に記載されるべき事項が後工程の設計書に記載されるケースも多い。そのため、上位の工程で決めるべきことが決められないまま後工程で作業されることとなり、結果的に手戻りも多くなり開発期間・開発予算が守れないリスクが後工程になって明らかとなり大問題となるケースも多い。

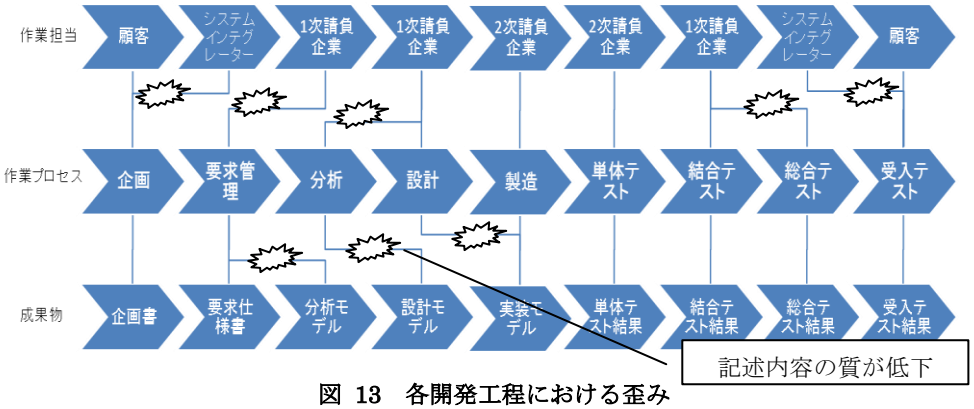


図 13　各開発工程における歪み

特に分析・設計以降の作業プロセスは“請負契約”によって、完全に 1 次請企業に任せることになっており、開発する中身をすべて任せてしまい、その内容（要求仕様を実現するためにどのような方式、やり方を採用しているのか）について関知しないケースも多い。

このように、本来の責任範疇である業務プロセスを十分に行えず、他社に任せてしまっているケースが多く、業務プロセスと業務担当に歪みが生じている。結果当然であるがその成果物に対する品質は十分なものであるとは言い難い。何とか帳尻合わせをしているというのが実情である。

参考に情報サービス産業に属する企業の委託状況を示す。このグラフは調査対象企業が該当する業務を全て選択した結果であり、このようにシステム化計画段階から要求分析（=要求管理）といった上流工程をアンケート対象となる企業の半分以上が委託を受けている状況にある。このように本来“開発（基本設計（=分析）以降）”が主な業務である情報システム関連企業の多くが上流工程の作業を委託され実施しているのである。

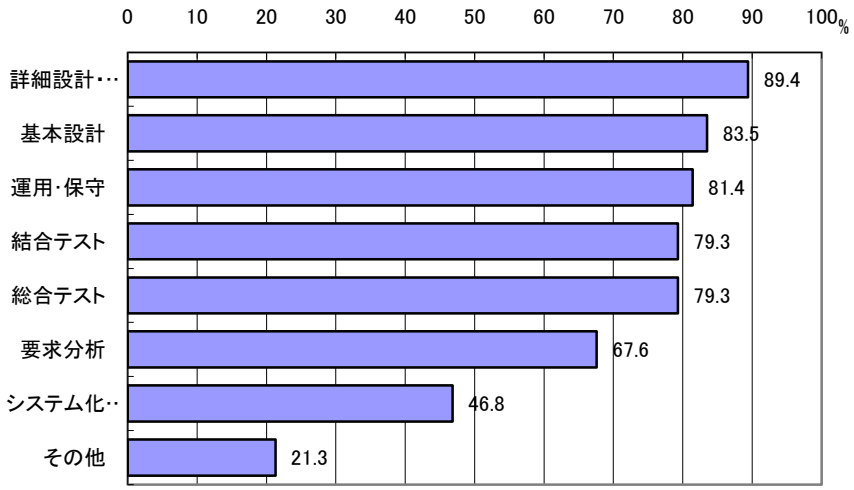


図 14　顧客から委託されている業務

出典　情報サービス産業白書 2009

次に動的なビューから、つまり過去から現在にかけての産業構造の変化に着目して IT 産業の分析を行う。まずは過去の IT 産業発展時のビジネスモデルの概観について以下に述べる。本研究では前述の定義では総務省で細分類上の受託開発ソフトウェアを主な題材として議論を進めるが統計データ上、まずは中分類である情

報サービス業を元にして全体感を掴むこととする。

下図のとおり情報サービス業従事者一人当たりの売上高推移で分かるとおり、2002 年前後で傾向に変化がみられる。前半は一人当たりの売上高も上昇傾向が続き推移している。しかし 2000 年を超えたあたりからはその上昇傾向の勢いがなくなり、横ばい傾向が続いているように見える。つまり、情報サービス産業の財務的な観点だけで判断すると、成長期から安定期または停滞期へ遷移しているのではないかと推論できる。2002 年からはまだ 10 年も経っていないことから、今後の傾向がこのまま続くのかもう少し様子をみる必要があるが、この定量的なデータと以降で述べる定性的な観点から、1990－2000 年前半までと、それ以降に分けてその概観を分析していく。

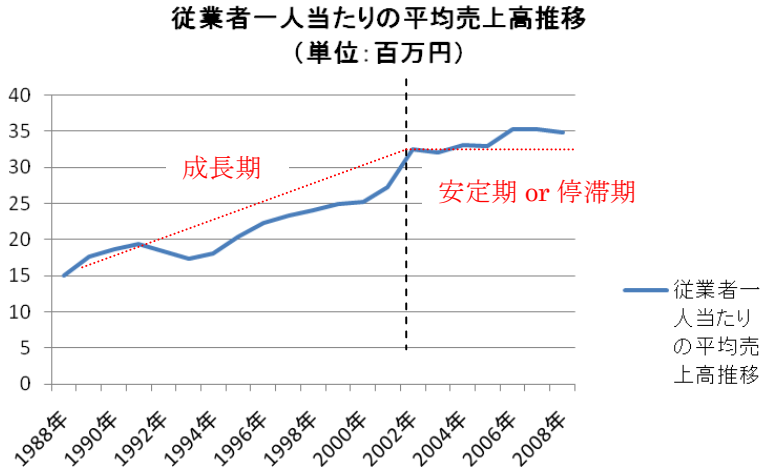


図 15　情報サービス産業従業者一人あたりの平均売上高推移

経済産業省　特定サービス産業動態統計調査（長期データ）より算出・編集

http://www.meti.go.jp/statistics/tyo/tokusabido/result/result_1.html

まずは前半となる 1990-2000 年前半までについて、その遍歴を考えてみる。各大手の企業は、BPR などという言葉を使って IT の導入による業務改革を積極的に行っていた。当然それに見合う IT 投資を行い、また IT 化案件を受注したシステムインテグレーターは自社の技術を用いて、そのシステム開発を行っていた。そのように IT 産業が発展している時代では、新規案件もまだまだ多く、保守による売上よりも新規案件による売上のほうが多かった。また、そのように色々な案件での開発を行う中で自社の持っている技術を洗練させ（品質が向上し）、次の受注に向けた準備を行っていた。結果的に自社の開発作業の生産性向上によるコスト低減や、お客様の新しい問題に対するソリューションを提言することに結びつき受注が拡大し、また実際の顧客も更に IT 投資を続け、顧客とシステムインテグレーターなどの情報システムの開発側とが良いサイクルで回っていた時代であった。つまり、この時代は IT 化に対する顧客の意欲そのものが業界全体を好循環に導くドライブフォースだったと言える。

またこの当時は現在のように様々な情報技術（IT）を持った様々な IT 関連企業があったわけではなく、システム開発に関わるステークホルダーが今よりも限定されており、開発方針などに今ほど色々な思惑が絡むことはなかったように考える。そもそもこの時代では、手作業が IT 化されるだけで十分であったため、その目的がシンプルで、その投資効果も得やすかったということも背景としてあるのかもしれない。

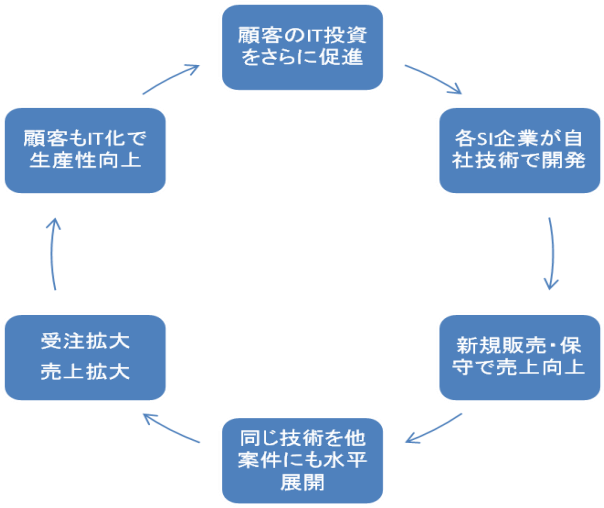


図 16　1990~2000 年前半までの SI 業界ビジネスモデル

しかし現在では IT 業界停滞時のビジネスモデルに変化してきており、2002 年以降顧客の多くが IT 化を終えた現在では開発したシステムの維持管理にかかるコストを如何に低減するか、ということが大きなトピックとなっている。結果的に前述のグラフのとおり、情報サービス産業従事者一人当たりの売上高も上昇傾向から現状維持ベースに変化を示していた。

つまり、システムインテグレーターから見ると、市場と考えていたパイの大半が IT 化を終えてしまい、今までと同じようなビジネスサイクルが回らなくなってきているのである。特に大手企業の場合、その基幹システムは一度開発し運用を開始してしまえば、随時に機能追加するにしても 5 年から 10 年は同じ情報システムが稼働し続けるため、そのリプレイスチャンスは当分なくなってしまうのである。例えば、日本オラクル社の決算状況などからもそのことは自明である。

以下 ITpro（2008）よれば「日本オラクルは 2008 年 12 月 24 日、2009 年 5 月期の第 2 四半期（2008 年 6 月－11 月）決算を発表した。連結売上高は 579 億 8800 万円（前年同期比 6.6%増）、営業利益は 176 億 7200 万円（同 0.4%増）と上期は増収増益を保った。しかし、世界的な景気減速の影響は大きく、同社は通期業績予想を下方修正した。　09 年 5 月期の売上高予想は期初予想より 90 億円下回る 1210 億円、営業利益は 2 億円下回る 392 億円になる見込み。売上高が予想より大きく下回る原因は、「データベースの新規ライセンスの販売減少が見込まれるため」と遠藤隆雄社長は説明する。　景気後退の影響により、設備投資の削減や見直しを行う企業が増えている。「アプリケーションやミドルウェアは着実に伸びている。顧客のコスト削減に結びつく即効性・実現性の高い提案をしていく」（遠藤社長）ことで、なんとか 09 年 5 月期も増収増益を達成したい考えだ。 」というコメントではあるが、日本オラクル社サイト_決済財務情報では下図に示す通りであり、「アップデート＆プロダクトサポート」事業が売上の一番大勢を占めている状況にある。企業経営としては安定傾向に入ったとも言い換えることもできるが、新規の成長が鈍化しているということが読み取れる。しかも、オラクル社と言えば、データベース製品が主力であり、どのような情報システムを構築するにも必ず利用される製品と言っても過言ではなかった。そのような製品に鈍化傾向がみられるようになったことは、オープンソース製品の利用率が高まったこともあるかもしれないが、新たに情報システムを構築する案件が減少してきている傾向を如実に示している。

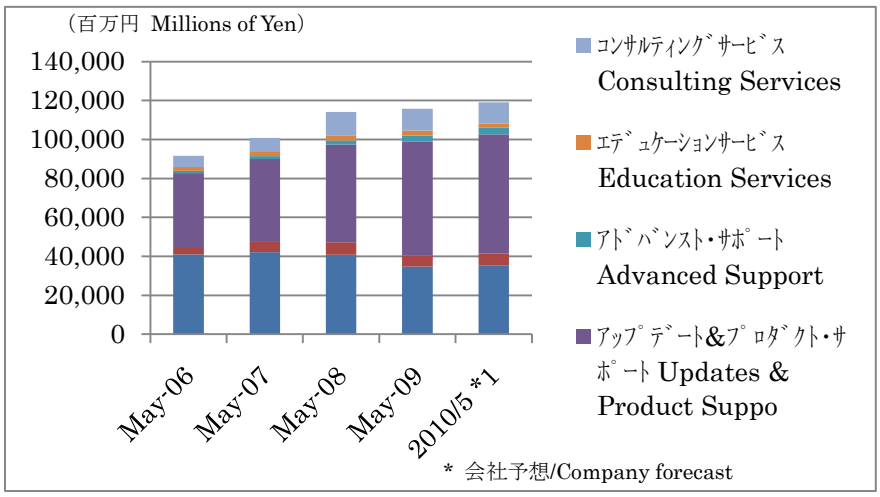


図 17　日本オラクル　ソフトウェア・サービス別売上高

Revenue by Segment / Results and Forecast

出典　日本オラクル社サイト データより編集　http://www.oracle.com/lang/jp/index.html.

事業	事業内容
ﾃﾞｰﾀﾍﾞｰｽ&ﾐﾄﾞﾙｳｪｱ Database & Middleware	リレーショナルデータベース管理システム、ミドルウェア製品群および開発・管理用ソフトウェアの販売
ﾋﾞｼﾞﾈｽ･ｱﾌﾟﾘｹｰｼｮﾝ Business Applications	CRM、EPM、ERP、IndustryApplication等を提供するOracleApplicationsの販売
ｱｯﾌﾟﾃﾞｰﾄ&ﾌﾟﾛﾀﾞｸﾄ･ｻﾎﾟｰﾄ Updates & Product Suppo	ソフトウェアプロダクトの更新権の提供、一般的な製品サポート、ならびに潜在的な問題の事前回避を可能とする技術情報の提供
ｱﾄﾞﾊﾞﾝｽﾄ･ｻﾎﾟｰﾄ Advanced Support	アウトソーシングサービス「Oracle On Demand」や予防的サービス「Advanced Custmer Services」党の高付加価値サービスの提供
ｴﾃﾞｭｹｰｼｮﾝｻｰﾋﾞｽ Education Services	技術資格の認定、システム技術者およびエンドユーザー向けのソフトウェアプロダクトの研修の実施
ｺﾝｻﾙﾃｨﾝｸﾞｻｰﾋﾞｽ Consulting Services	ユーザーのシステム構築に関する支援のための各種コンサルティングサービスの提供

図 18　日本オラクル社サービス内容説明

出典　日本オラクル社サイト データより　http://www.oracle.com/lang/jp/index.html

そのような状況から、結果的に現状でのビジネスサイクルは以下のような形に落ち着いている。各システムインテグレーターは開発費の低減、品質向上努力を懸命にしている状況である。そのため企業ごとに色々な技術開発を行ったり、また他社提供の技術をカスタマイズしたりするなどして、どのような技術が有効なのか検証・評価を行っている。そのような状況に対し、様々な製品メーカが売込みを行っている。生産性向上に効く技術や、将来の機能拡張へ有効な技術とか、色々な売り文句でシステムインテグレーターや、実際にシステムを利用する顧客にも売り込みを行っている。その売込みのため、毎年新たなコンセプトを作り、そのコンセプトを実現するための製品も提供する（SOA、Cloud などもその事例と言えるかもしれない。コマーシャル先行で、実際に宣伝通りの効果を得るための敷居は高い）。しかし、実際にその技術を活用して開発するにはある程度メジャーであり、活用しても問題がないのか（つまり新しい技術故に故障などの問題が何か潜んでいるの

ではないか、枯れた技術なのか）という不安を払拭してくれるものでないとミッションクリティカルな開発には活用できない。そのような経緯から、次々に色々な製品が開発されても本当にどれが良いのか判別できないまま標準化もされず、結果として、新しく生まれた有用な技術も実践に耐えうるレベルまでに洗練されることなく消滅してしまうこともある。よって、延々と各企業が個別に独自の考え方で生産性・品質向上に向けた努力を繰り返しているサイクルが続いており、業界全体としての生産性・品質向上のレベルが上がっていかず、顧客にも新しい提案ができないような状況になっている。

このように市販のソフトウェア製品に対しては顧客の立場となるシステムインテグレーターが、また開発した情報システムに対しては本当の顧客企業が、品質に対して不安を抱きながら活用すべきか悩みながら作業を進めている状況なのである。

顧客の IT 化意欲が低下した以上、過去のドライブフォースでは業界の好循環を作り出すことはできない。また社会的背景も変化しており、前述のとおり情報システムの社会インフラ化に伴う“品質“に対する重要度も上がっている。そのため今までと異なる業界のドライブフォースを見つけ出し、その力を上手く利用していかなければならない。

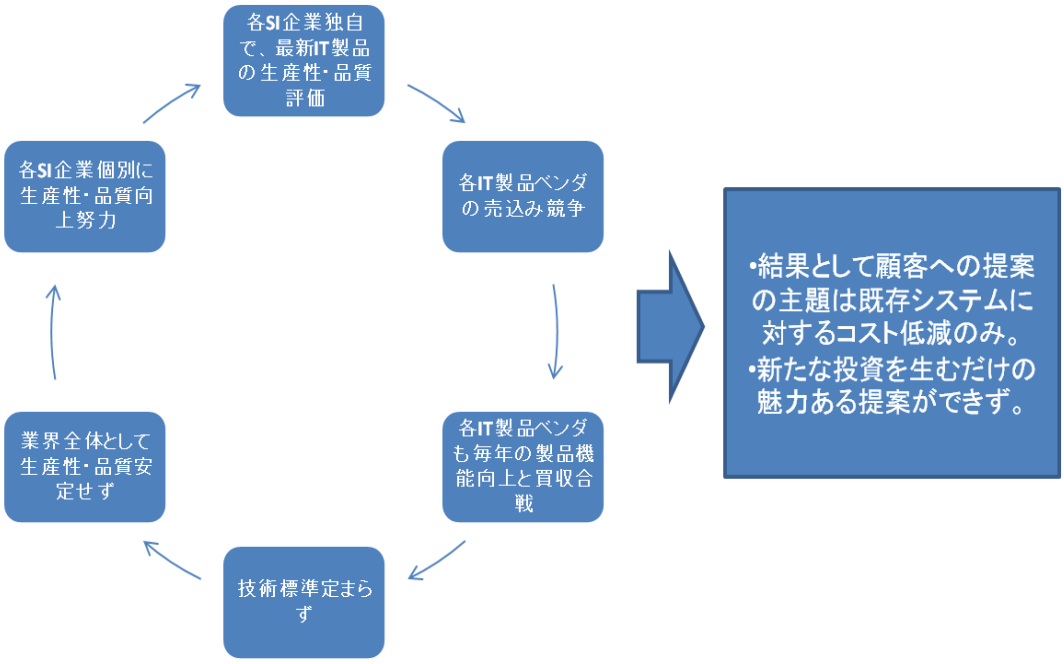


図 19　最近の SI 業界ビジネスモデル

しかし、このように顧客市場の拡大が出来ないまま次に述べるように最近では様々なプレーヤーが業界に登場している。そのために混沌とした状況に陥るのみで単純にそのドライブフォースを見つけ出すことは出来ていないのが現状である。

過去、汎用機開発の時代はどちらかと言えば、下記のような構造化された体制の下で情報がやり取りされていた。非公式には構造化された体制とは異なる情報のやり取りはあったが、多くの情報は統率された体制の下でやり取りされ、プロジェクトの意思決定も行われていた。しかし、最近では名目上の体制は構造化し整理された体制を整えているが、実質的な情報のやり取りは様々なルートが取られるようになっている。それは依然の非公式ルートが公式化されてきた状況に近い。それは、以前のシステム開発では汎用機メーカ 1 社でひとつのシステムを構築していたが、前述の通り市場は拡大していない状況の中で参加プレーヤーが増加し、各企業

が開発した市販製品を様々に組み合わせてシステムを構築する時代に変化してきた。そのため、ひとつのプロジェクトには依然よりも多くの企業が参画し、また各社の製品情報をそのプロジェクト内で共有するための情報流通も非常に重要な要素となっている。例えば市販製品の有用性を顧客に説明する必要もあり、また市販製品の利用方法、開発方法を実際に導入して開発する企業に対して説明することも必要である。このように、今まではプロジェクトマネージャ主体でそのプロジェクトの動きをコントロールすることが出来たが、市販製品を導入することにより、市販製品を開発し販売する企業（その企業は完全にプロジェクトのコントロール下にいるとは言えない）の動向もプロジェクトに影響を及ぼすようになってきたのである（製品同士の組合せ保証や製品保守の期限など）。

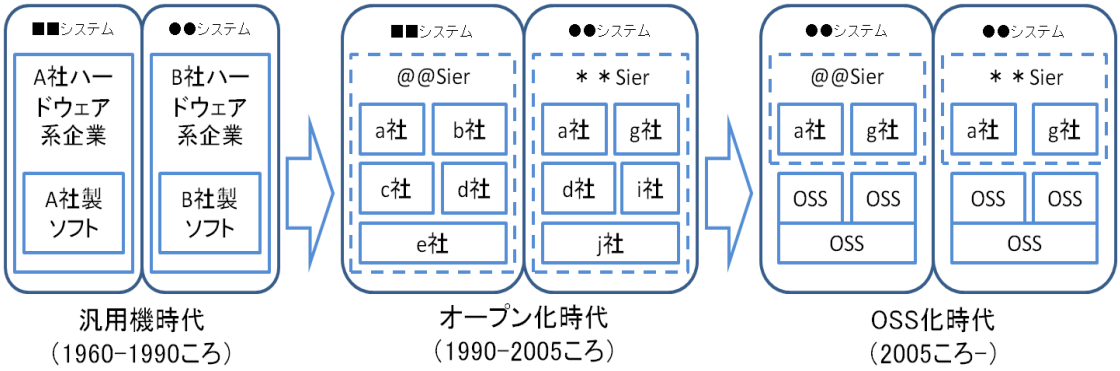
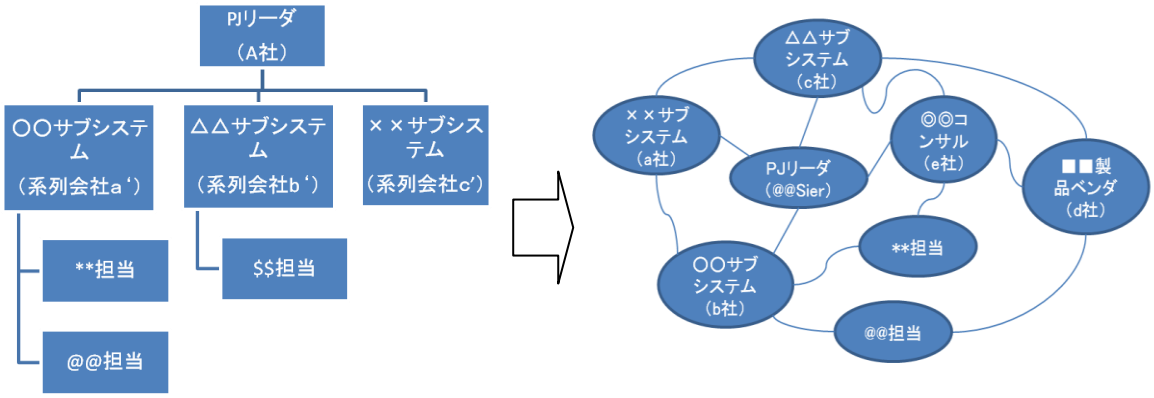


図 20　システム開発構造の変化

もう少し単純化したモデルで整理すると、1 社内に閉じた構造化した体制から複数社に跨ったネットワーク化した体制へ変化してきていると言うことができる。社会学的に言うならばある企業の系列会社を含んだヒエラルキー型の体制から系列に関係なく複数企業から成るヘテラルキー型の体制へ変化してきている。しかし、最近ではオープンソースソフトウェア（OSS）の利用が盛んになってきており、再度ヒエラルキー型に戻ろうとしているのではないか。オープンソース化は一般的には初期投資の低減効果（市販製品の購入コストがない）ことに着目されているが、誰でもが属せるコミュニティーが仕様を決めているが、ソースコードが公開されているため、その内容を理解できる技術者が自社内にいればメーカやコミュニティーに依存することなく修正などが可能になり、プロジェクト自体の独立性は高まる。このように OSS により各企業が絡み合ってひとつの情報システムを開発する体制から PJ リーダー主導のヒエラルキー型へ再帰する傾向があるが、本当に再帰できるかどうかはまだ微妙な段階である。というのも、結局 OSS 製品の保守はある企業へ委託することが多く、自社で保守まで対応しているところは少ないからである。そういったビジネスモデルが変化していくのであればヒエラルキー型への再帰も本格化するのかもしれない。

結局は現在のプロジェクトの多くはまだヘテラルキー化している段階であるが、その意識がないままヒエラルキー型のプロジェクト運営を行っている。ヒエラルキー型の運営では各作業者をルールで縛ることで運営が可能であった。設計者が拠り所とする設計ガイド的なドキュメントや、レビューと呼ぶ各設計者が作成した成果ドキュメントを他の設計メンバがチェックする作業などにより、ルール化した内容が守られているのか確認を行い、また当前であるが設計内容自体のチェックも行う。しかし、ヘテラルキー化してきてからは前述の通り基本的に上流工程で決まったことを下流工程にほとんど作業を任せてしまい、下流工程では正確に上流工程で決めた仕様を具現化しているのか、その内容の是非について感知しない。もう少し正確に言うならば開発期

間が短縮化し、開発費も低下してきていることから、下流工程の作業内容にまで入り込む余裕もないのである。まとめると情報サービス産業自体が安定期または停滞期に入り、様々なプレーヤーが参画するようになり、開発する情報システムの設計情報を流す対象が格段に増加している。情報が流れるルートも非常に多種多様な伝達ルートが出来上がり以前のようにプロジェクトリーダーが全てを管理できるような状況ではなくなっている。そのため、設計情報の制御が不能になりプロジェクトが混沌とした状況に陥ってしまうケースも多い。



ヒエラルキー型　　ヘテラルキー型

図 21　開発体制のモデル

構造化した開発体制から、ネットワーク化した開発体制に変化しつつあることを動的ビューの分析フレームワークを用いてライフサイクルに当てはめてその問題点を検討していきたい。

開発体制は“プロセス管理層”と呼んでいるところの議論に等しい。この最上位層がシステムの各フェーズにおいてどのように変化していくのか考えてみると、立上げ期では強烈なリーダーシップを必要とすることが通常であり、その組織構成は特にルールが定まったものでなくとも自然に各メンバの役割が決まって行くものである。次に成長期ではどうか。この時期もリーダーシップが明確になっていることは変わりないが、ある程度のルール付けが必要となってくる。このような時期には構造化し体制が明確になり、そのとおりに情報が組織内を流通していくことが全体を上手くまわしてくためには必要である。安定期になれば、ルールも緩めることが可能であり、厳密な規則を作らなくても各メンバも習熟していることから大抵上手く回ってく。また組織構造も厳密に構造化されたものでなく、非公式な情報のやり取りなども活発に行われ、言ってみればネットワーク化した組織に変化してくる時期である。衰退期では、組織は小規模化するかまたはネットワーク化が更に進化してハブ的な役割を担う人も居らず、混沌とした状態に陥って行くケースが多い。

以上が形式的なプロジェクトのライフサイクルモデルであるが、情報サービス産業ではどうであろうか。はじめに述べたように、近年情報システムで社会問題化したトラブルが多く、そして統計データからも多くの問題が内在していることが分かっている。しかし産業全体の売上高から見ると安定期に入ってきたと考えられる。通常の場合、安定期ならば品質は安定しており、少なくとも社会問題化するほどのインパクトのある問題は少ないはずである。そして安定期であるが故に参加プレーヤー間の情報伝達がネットワーク化していることは想定されるべきことである。現実にはカオス化し結果的にはシステムの品質も悪い、これらは元々の情報伝達方法が確立していないまま複雑なネットワークにより設計情報をやりとりしなければならないような状況に陥

ったこと、つまり参加プレーヤーがオープン化により急激に増加したことで必要な情報が必要なプレーヤーに正確に、またはタイムリーに届かない事象が発生していることを意味していると考えられる。

これらを踏まえて、今の品質・生産性を高めるための仕組みをどのように考えればよいのであろうか。実際に情報システムを開発するプロジェクトは、通常では開発開始時に寄せ集めでスタートし、開発が終了すれば一部の技術者を残して解散してしまう。このように一時の集合体に過ぎないプロジェクトでは、開始時にネットワーク化した情報網を基盤にしていては統率を取ることが非常に難しい。個々の情報システム開発プロジェクトをコントロール可能な状況にするには、ネットワーク化した組織の中で情報のハブになることが必要条件となる。プロジェクトメンバの属人的な能力に関係なく、常にそのような状況を作り出すにはどのようにすればよいのか、それが現在のプロジェクトマネジメントの課題ではないだろうか。プロジェクト内でのルールを厳密にして、それを厳密に守ることを課すことは安定期に入った産業にとっては必ずしも良い選択とは言えない。ルールを厳守させるためのコストが増大し、またルール厳守の手間のために開発効率を下げる要因にもなる。つまり、全てをドキュメントで規定し、それぞれの作業を常時チェックするような厳密なルールよりも適切な“ゆるさ”が必要なのである。金融業界で言われている「ルールベースからプリンシプルベースへ」的な発想がプロジェクトマネジメントにも必要であろう。

慶應義塾大学大学院高野研一教授の講義「ヒューマンファクター」でも説明されているが、分厚い規定を作り、従業員がそれを覚えるだけでは出荷される製品の品質を担保する十分条件にはならないし、組織は上手くまわらない。雪印でも HACCP（Hazard Analysis and Critical Control Point）を導入し、認証を得た後に毒素混入事件が発生し新聞に大きく取り上げられた。細かく規定された手順を準拠することで一見正確な作業をしているようには見えるが本質的なところで漏れがあったり、また認証を取ることだけが主眼となり実務レベルに定着しなかったり、と実社会では雪印に限らず多くみられる事象である。

情報を円滑に伝達し、適切な判断・アクションを可能とする仕組み化が大きな課題であると言える。

6.　日本の IT 産業に見る情報のカオス化による影響分析

前章での分析を踏まえて、日本の IT 産業の問題点を整理する。前述の通り、業務担当と実際に行う業務プロセスの対応、そして業務プロセスとその成果物の質に歪みが生じていること。そして、近年プロジェクトに関わる情報の流通がネットワーク化しており、マネジメント担当が全容を把握し切れないまま、周辺の事情でプロジェクトに関わる様々な事柄が決定してしまうこと。この２点が日本の IT 産業を取り巻く大きな流れとなっている。

静的分析、動的分析を行う中で明らかにしたこととして、下記の統計データからも明らかなように産業に関わるプレーヤーが近年増加し、そしてひとつの情報システムを作るときに関連するプレーヤー（企業）も増加している。大規模な情報システム開発案件が減少傾向にあるにも関わらず、プレーヤーが増加傾向にあることは、ゼネコン的な性質もさらに強くなり、自社責任作業を委任契約・請負契約により関連企業へ流している傾向が高まっていることの結果とも推測できる。

この問題構造を考える上で、ただ単純にこの流れ自体を止め、すべて自社開発にするように方向転換すればいいのではないか。というアイディアが浮かぶかもしれないが、それは非現実的なことである。たとえば大規模なシステム開発のように自社の要員だけではまかないきれない稼動を必要とする場合には、やはり作業を分担できる下請け企業は必要不可欠なものであるからである。よって、分業化することを前提にして如何に品質・生産性を向上させるべきか、ということを考えるべきである。

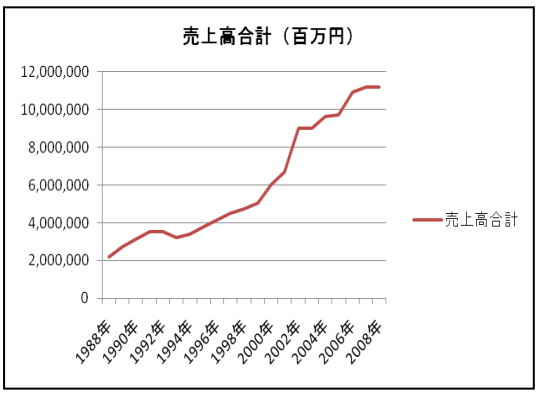


図 22　情報サービス産業売上高

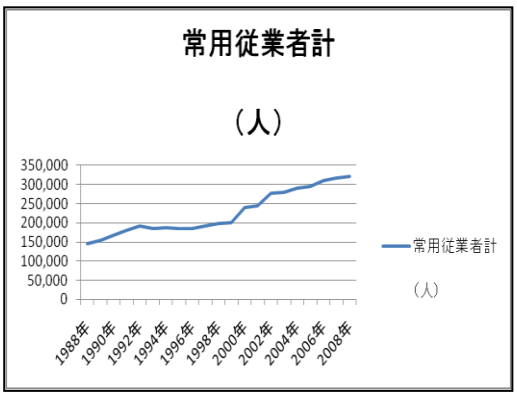


図 23　情報サービス産業の事業所数

経済産業省　特定サービス産業動態統計調査（長期データ）より編集

http://www.meti.go.jp/statistics/tyo/tokusabido/result/result_1.html

では、以降では“作業責任の歪み”と“情報伝達ルートのネットワーク化“によって、各作業プロセスで何が起きているのか詳細に見ていくこととする。

◆　各業務プロセスの状況について

・　企画プロセスについて

業務プロセス上、顧客自身で行うべきことをやり切れず、システムインテグレーターが本プロセスに大きく関わりだしている事象について述べる。

情報システムへ投資を決定する顧客の経営層は、安い賃金でオペレータを雇い、且つお客様には高い品質のサービスを提供したい。そのため、自社のユーザは単価の安い派遣社員に作業を任せるが、高いスペックの情報システムで補うことで高い品質のサービスを自社のお客様へ提供することが狙いであると考えている。

しかし本来は、このように情報システムという“手段“が、はじめから問題解決手段の前提として決まっていることは、その解決策の選択肢を狭めることになり、また問題として定義付けた内容も、それが本質的な事柄とは異なってしまうケースも多く、好ましいソリューション検討であるとは言い難い。

“手段ありき”のソリューションは部分解だけにしかならないケースも多く、本質的な解決策にならないため、経営層が狙う効果が得難い。

つまり、企画段階でシステムありきのソリューションしか持たないシステムインテグレーターはそのソリューションを限定し、狭い領域での選択肢を顧客に提示することしかできない。近年システムインテグレーターがコンサルティング会社を買収するなどして、協業しだしたことでその傾向は多少修正されてきているのかもしれないが、IT への投資効果の評価が厳しく問われるようになっている現在、そのソリューションを手段ありきでのものではなく、様々な観点から検討し、現実性や即時性、コストなどを考慮して情報技術（IT）を用いたソリューションが最善である。というような思考で顧客へ提案することが必要だと考える。

安易な解決策の策定は情報システム化において失敗を招くし、安易な提案はその投資効果として何を評価すればよいかわからなくなってしまうケースが多い。

また、この段階では各製品メーカが顧客、そしてプライムコンストラクター、1 次請け企業などへ売り込み、自社製品の導入を働きかけるフェーズである。採用を検討しなければならない製品はひとつではないため、互いの連携が可能なのか、またその製品を使った開発が可能な技術者がいるのか、など多方面への情報収集、情報のやり取りが必要となる。

・　要求管理プロセスについて

要求条件を決めるフェーズであり、全体のプロジェクトマネジメントを行うシステムインテグレーターが抑えるべき工程である。どのような要求条件を何故実現しなければならないか、それらのことを把握することは、プロジェクトの予算管理、スケジュール管理、品質管理などに影響する。しかし、このプロセスの実質的作業を 1 次請負会社に任せてしまうケースも多く、結果的に自力のみでの予算管理やスケジュール管理が難しくなっている。最後は実作業を行った 1 次請負次第なのである。

このフェーズでは情報システムの機能に対してどのような要求があるのか整理し、そして何故そのような要求条件の実現が必要なのか、ということまで明らかにしておく必要がある。しかし、多くの場合はどのような仕組みで要求条件を実現するか、という観点に重きを置いてしまい、要求条件が出てきた背景まで明らかにすることは少ない。要求条件の背景まで明らかしておくと、システムを開発する側からも他の推奨ソリューションを提案することもでき、顧客側はそれらの提案を取り入れて幅広く検討して要求仕様を決定することができる。背景がわからないままでは、ソリューション検討の深みが増さないのである。

結果として、安易な要求分析の場合は本当に改善すべき問題が曖昧であり、実は IT 投資をせずとも他解決策があるようなケースも多く、また過剰な要求スペックを情報システムに作りこむことにもなり、その後の保守・運用費を高止まりさせる要因にもなるため注意が必要である。

・　分析・設計プロセスについて

多くの場合にシステムインテグレーターは、「請負契約」で 1 次請け企業に作業を丸投げすることで、開発出来なかった場合のリスクをその請負企業に背負わせてしまう。「委任契約」ではないため、納品時に納品物の確認はするものの設計内容の詳細について十分に理解はできておらず、概要レベルしか把握していない。結果として非常に乱雑な仕組みで要求条件を実現してしまうケースもあり、開発完了後の保守が非常にやり難くなっていることも多い。各部品の独立性を高めてそれらの部品を組み合わせて作ることができていれば、仕様が随時変更していくときにも、ある程度の変更容易性を確保することができるのだが、多くの場合では独立性を高めた設計を検討するよりも、必要と考えた部品を思いつくレベルで作り込んでいくことを選択してしまう。それは 1 次請け企業が開発作業の中で非常に負荷がかかる役割を担っていることもあり、十分な設計検討時間を取り難いということも理由のひとつである。2 次請企業に仕事を振ってしまい、着手できる部品から作りこんでしまってもらったほうが期間の決まった開発作業の中では適切な判断なのである。

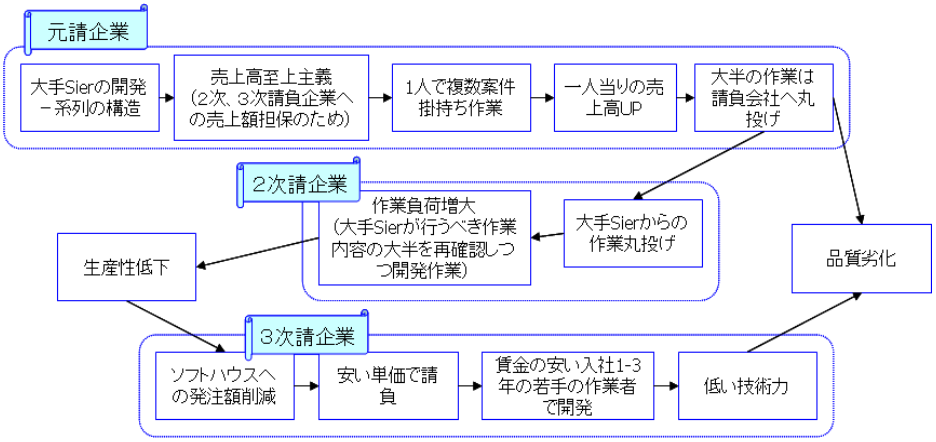


図 24　情報システム開発プロジェクトのゼネコン構造とその問題構造ツリー

また、要求管理プロセスで述べたとおり、1 次請け企業はピーク時には要求条件を決めつつ分析設計プロセスも行うことになり、全体としての負荷が非常に重い。

・　製造作業について

下流工程の技術者は、プロジェクト内では請負企業の実施する作業であり、その中でも地位が低く見られる傾向がある。プログラマーといえば一般的に低賃金で過酷な労働を強いられることが通例である。そのため 30 代前半くらいまでにはプログラマーを卒業し、上流工程の開発担当へ移っていくことが一般的なキャリアパスとなっている。結果、入社して 2-3 年程度の経験の浅い開発者が主体のコーディング（ソースコードの作り込み）作業となっているが、そのようなレベルでは学生のプログラミングの域を脱しておらず、まずはコーディング規約を守ってもらうことから教育を始める必要がでてくる。特に、製造工程で必要な作成したプログラムの原本管理（構成管理・変更管理）については、学生時代には全く経験のないことで、その重要性から理解を求める必要がある。

当然、非常に優秀な人材もいるが、大半はこのようなレベルの作業者が社会インフラを担うような情報システムを直接動かすソースコードを書き、実行するのである。数百万行にも及ぶソースコードをそのような技術者が作成し、どうやって品質を確保すべきなのか。プログラマーを低い地位としてみなすのではなく、もっと

重要な職種としての処遇と優秀な人材の育成が必要なのではないだろうか。

また、上記と同様であるが、ソースコードレベルになると、技術者と言っても殆どのシステムインテグレーターは品質レベルを自力で判断できなくなる。50 代の技術者の方々は過去に COBOL などのプログラミング経験があり、最新のプログラミング言語でも勘が働くところはあるが、システムインテグレーターで要求管理などの仕事しか経験のない 20-30 代の技術者では製造経験のない技術者も多い。請負企業の若手は上記で紹介したとおりプログラマーから始まって、プロジェクト管理を行うように成長していく。しかし、上流工程のみを業務とするシステムインテグレーターはそのようなキャリアパスを通ることなく、ある年齢になるとプロジェクト管理を行うようになるが、技術的な信用度という面では請負企業のプロジェクトマネージャの方が高い信用度があることのほうが多いのではないかと考える。というのも、単にプログラミングの経験有無のみを問題視しているわけではなく、上流工程のみしか経験していないマネージャーは、プロジェクト全体で各作業工程ではどのような役割の担当が何をしなければならないのか理解していないため、また分析設計の箇所でも記述したが、どのような仕組みでシステムが動いているのかも分からないため、マネジメントも表面的になってしまっており、また問題が発生してもそのときどのように立ち振る舞うべきか分からないのである。

・　単体テスト・結合テスト・総合テスト

テストを行うためには何をテストすべきかそのテスト項目を抽出し整理することが初めの作業となる。しかし、その作業は（設計書、ソースコードから自動生成するケースもあるが）多くの場合は技術者のスキルに依存する。他の技術者によるレビューなどを行ってはいるが、短期での開発では十分にそのような第 3 者チェックが行われているとは限らない。そのためテスト内容が充足しているのか否か曖昧なケースも多い。各システムインテグレーター、請負企業独自に基準は決めていることが殆どであるが、形骸化している基準である。（但し、組み込み系については、テスト手法が発達しているようであり、その品質管理は情報システム系よりも厳密に行われる傾向が強い）

テストが物理的に困難な場合もあり、実際のサービスを開始するまで発見できない故障箇所も存在する。それは本来であれば開発作業中にテスト工程まで考慮した設計を行うことで、そのようなケースを避けることができることもあるが、前述のとおり作業を丸投げしていることもあり、テスト工程での作業まで見切らないまま設計しているケースが殆どである。

・　その他の業務プロセス

✓　コンサルティングについて

最近はシステムインテグレーターや PKG ベンダが顧客企業へのコンサルティングを行っている場合がある。その場合は、前述のとおり情報技術（IT）ありきでのソリューション提案になっており、本質的な問題とは別な解決策となっているように考えるケースもある。

✓　パッケージ（PKG）製品の提供について

PKG 製品は多くのシステム開発で活用され、多くの開発技術者がその製品の特性を習熟しようと努力しているが、その製品自体の品質には問題が多く、発売前に品質が安定するまでバグ潰しをするようなケースは稀で、販売した製品をシステムインテグレーターに採用してもらい、その技術者たちが開発工程中に発見する製

品のバグを修復していくことで品質を上げていくやり方が通例となってきている。

但し、製品の故障が発見されても、改修に時間を要するケースも多く、暫定的な対処のままお客様に納品することもある（これはオープンソース化の流れを作り出した要因のひとつでもある）。

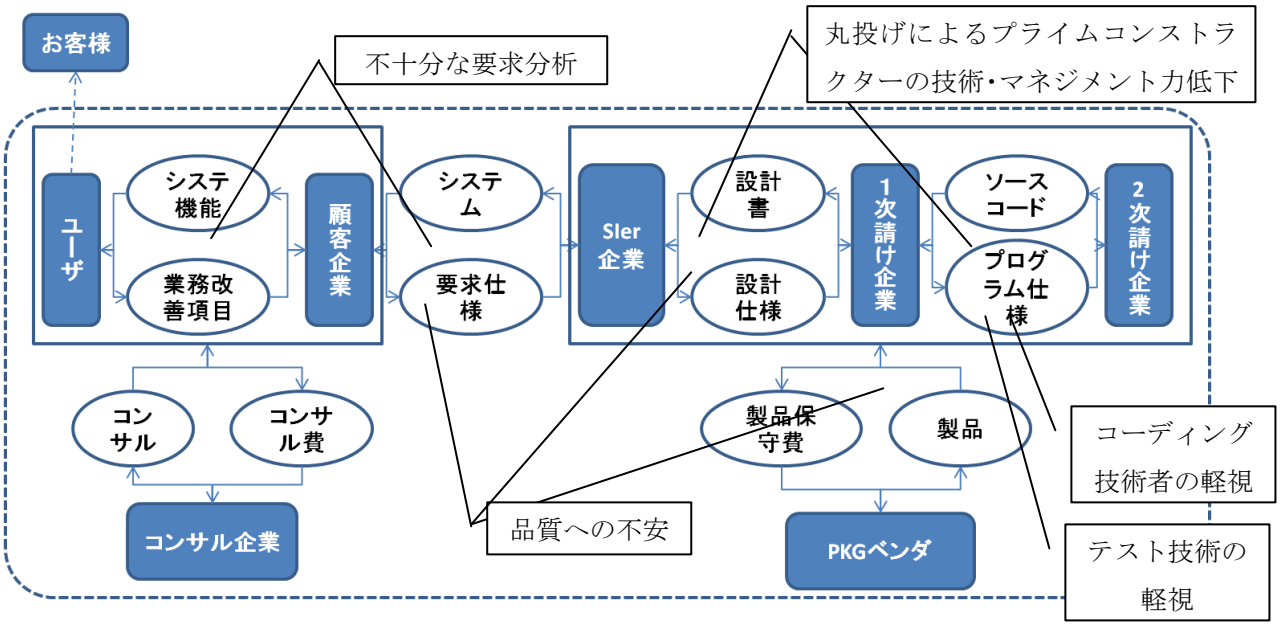


図 25　日本の情報システム開発に関わる問題事象

このように分業による歪み、情報伝達の複雑性による影響を実際に開発工程に則してみてきた。やはりその影響は随所に確認できた。

7.　問題点のまとめ

以上、これまで述べてきたように、近年では担当すべき作業を下請け企業に任せ、自分の本来やるべき仕事をしていないこと。そして、情報システムを構築するには昔のような構造的な組織体制で分業化して開発を進めるのではなく、複数の関係者とネットワークのように情報をやり取りする中で作業が進めていくことが多くなってきた。上記でみたとおり、それによる影響が様々な問題に派生し、プロジェクトの品質に影響を及ぼしている。プロジェクトをコントロールするためには情報の“ハブ”を抑えないと、混沌とした無制御なプロジェクトとなり、その作業品質の劣化の危険が潜んでいるのである。

＜本研究で解決すべき問題点＞

「本来すべき仕事ができていないこと」、「ネットワーク化による複雑化した情報流通」が現在の IT 産業の問題、特に情報システムの高品質化を考える上での課題と考え、その 2 つの課題から見える本質的に要因は「円滑に情報が流通されないこと」である。”Just in Just Time & Correct”に情報が流れる仕組みを整えることが命題なのである。

請負企業に任せることで本来すべき仕事をしていない中で何が問題になるかというと、例えば任せた仕事に対する要求を発注側が請負側に正確に伝えられていないため何度も手戻りが発生してしまいコストまたは納期超過が起きてしまうことや、または、請負側で行っている作業内容の詳細が曖昧なまま発注側に報告され、その作業の質自体をしっかりと管理できないため、結果として仕上がった作業が非常に低品質なものとなってしまうようなこともある。このように情報のやり取りが発注側・請負側、または作業者の間で十分にできていなことが問題なのである。また複雑化した大規模プロジェクトではやり取りすべき情報も伝えるべきところに伝わらなかったりするケースもある。そのようにやはり「情報が円滑に流れないこと。情報が滞ること。」が本質的な問題なのであると考えた。

8.　情報の滞留原因

◆　そもそも情報はどのように伝わるのか

情報を伝達には、作業者はもちろん、その作業プロセス、そして開発するシステムを構成している要素（設計書など）も関連している。これら３つの要素が影響し合って伝わっていく。この場合の情報とは“設計情報”と同意と考えるとイメージし易いかもしれない。

作業者は、ある作業プロセスに応じて設計情報を加工編集し、設計書などの成果物（構成要素）に情報を残す。その情報を別の作業者が作業プロセスに則って、また別の加工編集を行う。そのような作業プロセスを通して設計書などの成果物というものを使って情報を伝達し、最終成果物まで仕上げていく。また、左記に述べた範囲はモノづくり工程のみであるが、顧客の声を聞き、製品仕様まで落としてくるまでも同様なプロセスがある。ニーズを見つけ出し、それを基本要件としてまとめ、製品の要求仕様までまとめることも、やはり作業者がある作業プロセスに則って成果物として情報をまとめ、そして伝えることで成立するのである。
開発される情報システムの品質は当然この伝達される設計情報に依存しているので、その品質も上記３つの要素に依存して決まる。よって、下記の方程式が成立すると考えた。

<u>情報システムの品質</u>
＝“システムを構成する要素（利用する技術、利用する入力情報（設計書など））の品質”
×“作業プロセス（作り方）の品質”
×“作業者の能力（技術力・コミュニケーション力 etc）”

では、高い品質の情報システムを作るためにはどうすべきか。

「システムの構成要素の質、作業プロセスの品質、作業者の能力」、これらの要素の質を上げることである。各要素の質を決定付ける変数は、システムを構成する要素ならば、ニーズ、基本要件、要求条件機能、部品、成果物、がそれに当たる。また、作業プロセスの質は作業、そのものであり、作業者の能力も、作業者の質である。これらの変数がどのような値をとり、そして各変数の関係性がどのようなものなのか、によってそのシステムの質は決まる。つまり、変数が適切な値を満たし、そして変数同士の関係性も適切なものであれば、情報は円滑に流れるのである。

◆　これまでの議論の整理とソリューション検討に向けて

情報システム産業を高い品質にすることで、安定した社会インフラを構築し、日本社会の質を向上、または日本経済の発展に貢献できるものにしたいということが最終的なゴールである。

また、現状分析の中で導き出したとおり、要求条件の曖昧さや利用する技術・製品の品質が悪いこと、そしてゼネコン構造による丸投げ体質、技術者のスキル低下の問題を解決しなければならない。現状、情報システム産業のゼネコン体質など表面的に見える問題は前述のとおり幾つもあるが、結局何が起きているのだろうか。それは本来すべきことをせずに仕事を他企業に任せてしまうこと、またそのように自社以外にも仕事を任せることで色々なステークホルダーが開発に参画することになり、その情報流通が複雑なネットワークと化しているのである。そのときに本質的に問題になることはシステムを構成する各要素に必要な情報が円滑に流通しなくなることである。そのように情報が滞ることは、たとえば必要な設計情報が伝わらずに故障を引き起こす原

因にもなり、非常に根本的な問題を持っている。

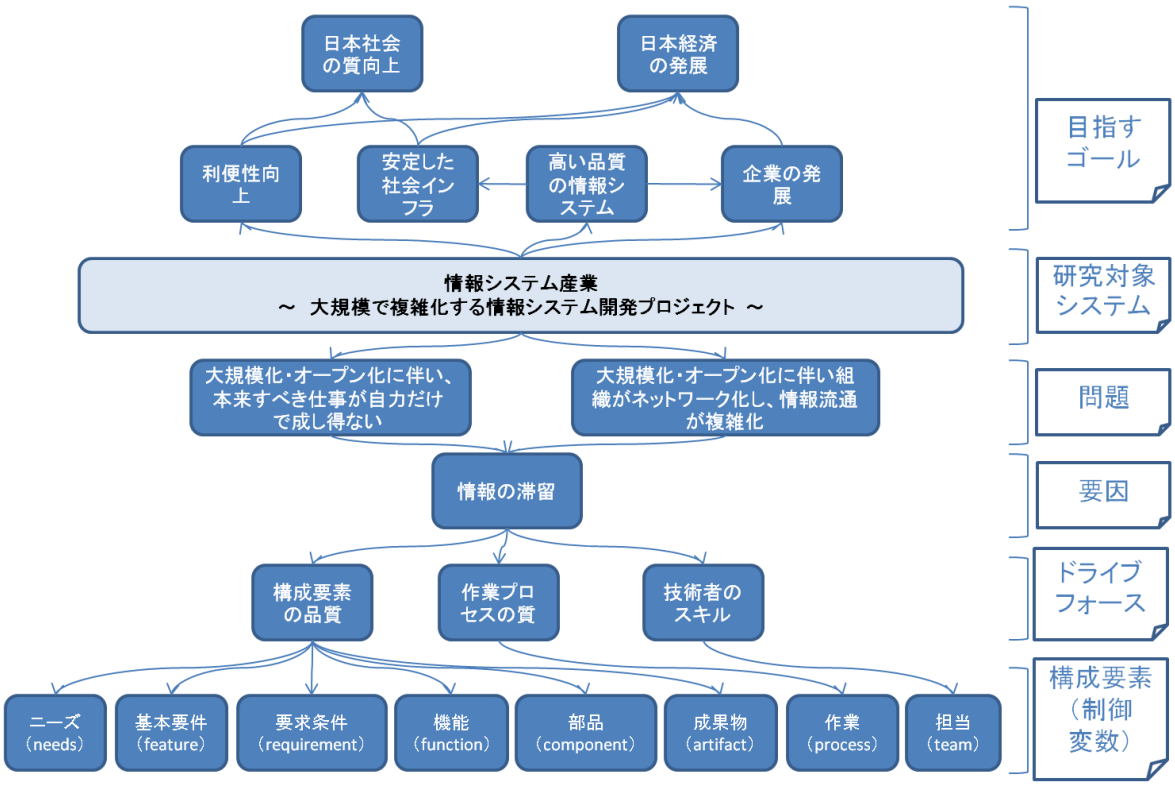


図 26　情報システム産業を取り巻く状況の整理

（これまでの議論の整理とソリューション検討に向けた本研究の全体概念図）

そこで、情報が滞るメカニズムを考えてみる必要がある。つまりシステムの質は何で決まるのかそのドライブフォースを明らかにし、それに対して対策を打っていくことが大切である。そこでシステムを構成する要素に何があるのか着目し、その組み合わせ方により全ての品質が決定付けられると考え、要素を洗い出した。大きくは３つの要素を抽出した。これは当初述べた静的分析のフレームワークでも説明したが、システムを動かす技術者（人・組織）の要素、また人・組織が作業するプロセス（タスク・アクティビティ）という要素、そしてその結果生成される成果物（最終成果物だけでなく途中成果物（設計書など））の３つである。これらの質がシステムを定義つける源となっているのである。

洗い出した３つの要素をさらに具体的に考えたものが最下層で示した８つである。ニーズ、基本要件、要求条件、機能、部品、成果物、作業、担当という８つのパラメーターこそが、実際にシステムの質を決定付ける要素なのである。（これは別の見方をすれば、ある組織活動をオブジェクト指向分析し、その組織を構成するオブジェクトを抽出した、というように説明することも可能である。）

以上がこれまでの議論のまとめであり、これを基本に以降のソリューション検討を進める。

◆　ソリューション・フレームワーク　～メソドロジー～

まずは、難解でコントロール仕切れなくなりつつある現代のプロジェクトのモデルを説明する。そして、モデルを用いてあるべき姿を解説し、どのような観点で組織体をシステムとして捉えれば良いか、そのポイントを述べ、モデルの有効利用方法を説明する。

ソリューション提示にあたって、システムへのインプット（市場ニーズ）からシステム全体が人（組織）、プロセス、成果物（プロダクト）の３つのサブシステムで構築されていると考え、その３つに分割して設計を行う。

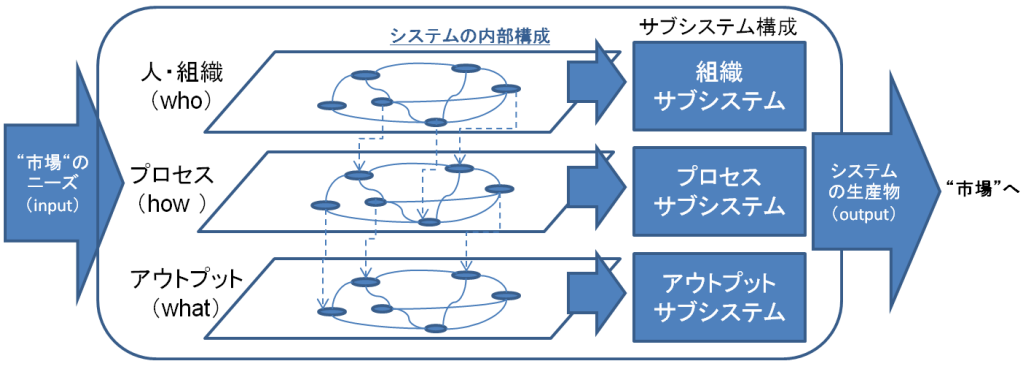


図 27　システム構成

また、各サブシステムは以下のような関係性を考慮して論じる。全体の関係性について論じられた先行研究は無いようだが、個々の要素関係について論じたものが MIT NamPyo.Suh(2004)、INSEAD_Manuel E.Sosa(2003)、MIT_Steven D.Eppinger(2008)などの論文で紹介されている。そのような先行研究で論じられているアイディアなどを適宜織り交ぜて議論を展開していくこととする。

そこで本研究で考えたモデルの構造について説明するが、まずは市場のニーズからサービス・製品として実現し提供するときに必要となる要素の関係性を整理したモデルを提案する。このモデルはニーズをサービス・製品化する過程で必要なキーファクターを洗い出し、その要素間の関係をマトリックスで整理したものである。（ネットワーク図で要素間の関係性を示すことは可能であるが、数値的なシミュレーションを行うことまで考えて、その関係性をマトリックス（行列）として表現することとした。）

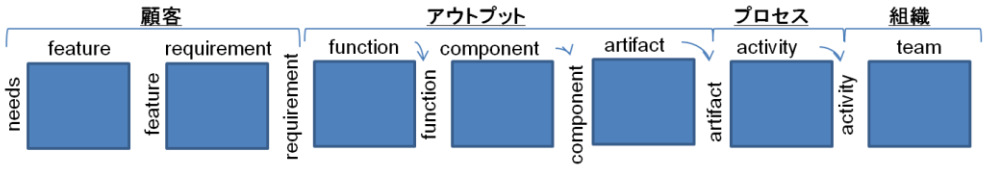


図 28　情報伝達の可視化モデル

市場のニーズ（needs）は基本要件（feature：システムが実現したい振る舞い（ex. システムは△△を実現すること）を定義）に詳細化され、基本要件は要求仕様（requirement：システムが行う振舞い方（ex. システムは〇〇を行い××となること）を定義）により定義される。この市場にあたる箇所は要求工学や、価値設計工学という分野で研究がなされている分野である。

また needs と feature、requirement の関係などと同様に要求仕様（requirement）と機能（function）の関係を示すマトリックス、機能（function）とそれを実装する構成要素（component）、構成要素（component）と成果物（artifact）、成果物（artifact）とそれを生み出す作業（activity）、作業（activity）とそれを担当するチーム（team）、以上それぞれのマトリックス関係を検討することにより、プロジェクトの状況を明らかにする。

現実社会においては、上記マトリックスももう少々複雑化させなければならない。例えば、組織について考えてみれば、所謂“管理組織“のようなものが存在している。つまり、あるチームをまた別のチームが管理し

ていることになり、チーム vs チームの関係が生まれている。そのような場合は上記マトリックスに組み込む必要がある。また別の例ではある機能を複数の機能により実現する場合などは requirement-function のマトリックスと function-component のマトリックスの間に function-function の関係を示すマトリックスを組み入れる必要がある。このようなやり方は開発の初期に本マトリックスの関係性が十分に明らかにできていないときにも有用であると考える。

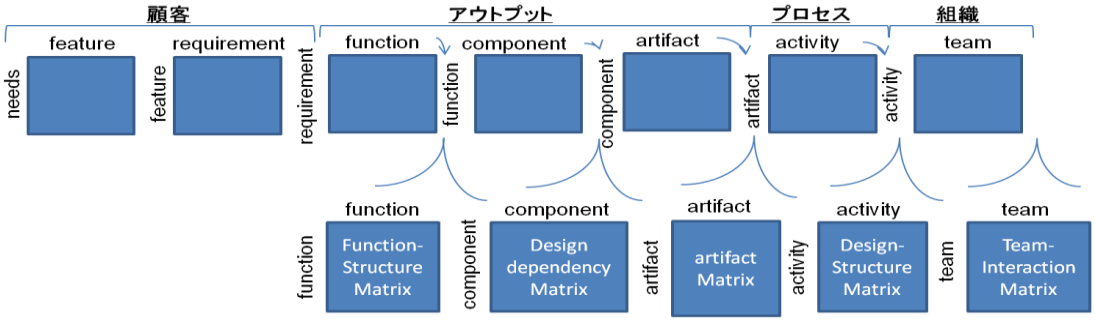


図 29　情報伝達の可視化モデルの応用

ソフトウェア開発の場合、大抵は開発当初に洗い出される機能は非常にざっくりとしたものであるが、設計工程が進むにつれて詳細機能が明らかになる。そのような場合に、開発当初は大きな機能（function）でマトリックスを作成しておき、開発工程が進むにつれて詳細機能が明らかになれば function-function のマトリックスを挿入することもやり方のひとつである。

次に本モデルに関わる先行研究との関係について述べる。マトリックスで整理し、改善を行う手法は DSM を始め、いろいろなものが研究されている。

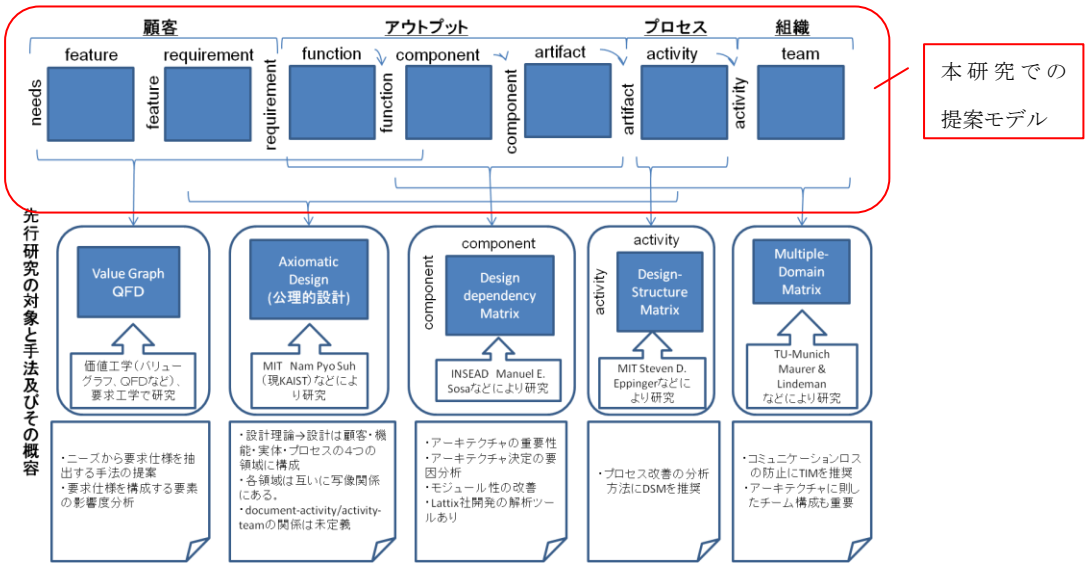


図 30　提案モデルと先行研究の状況

代表的な DSM は MIT の Steven D.Eppinger などにより研究がなされ、1990 年代から盛んに論文が発表されるようになった。そこでは主に製品開発プロセスの効率化が主題となるテーマであり、以下にその基本的な考え方を解説する。

＜DSM について＞

DSM とは「Design Structure Matrix」の略であり、下図（作業フロー図）に示すようにある作業のフローをマトリックス表現し直したものである。横軸が先行作業であり縦軸が後攻作業である。

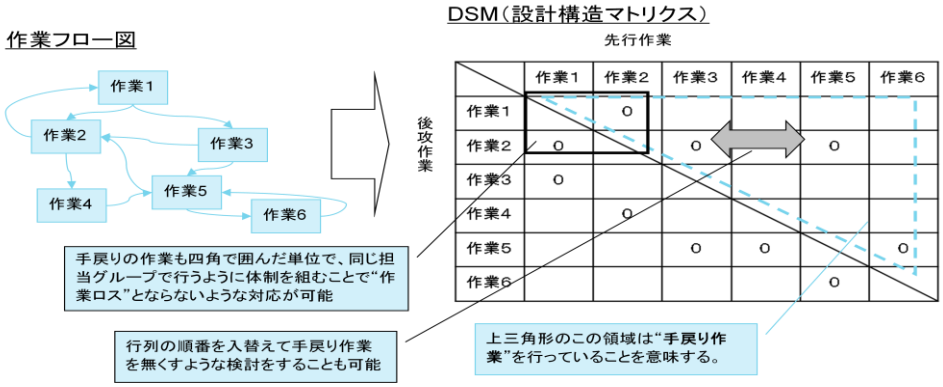


図 31　DSM とは

例えば“作業１→作業２”の順序を表現すると対角線より下側の作業１と２の交わるカラムに○が付くことになる。このルールから、対角線に対して右上にチェックされた印は手戻りを意味し、改善が必要な作業のひとつとしてみなされる。DSM 上で作業の順番を並べ替えたり、または作業を合同で行ったりすることで、たとえ手戻りがあっても対角線の周辺で作業が済むように全体設計を行い、作業全体の効率化を図るように作業プロセスを再設計することに役立つ。

また、この DSM は作業プロセスの関係性を整理するだけでなく、製品を構成する部品の依存関係を整理することにも用いられる。更にその部品設計担当チーム間の関係性について、この DSM が用いられて研究もされている。INSEAD の Manuel E.Sosa などが MIT の Steven D.Eppinger と共同で研究し、ダイヤモンド社ハーバードビジネスレビュー（2008）にも紹介されている。この場合は DSM とは呼ばずに TIM（チーム・インタラクション・マトリクス）という呼び方をしているが本質的なところは DSM と同じである。

更に最近ではマトリックスを利用した分析方法も様々な要素関係の分析に活用されるようになり、DSM も一般的な総称に近いものとなっているようだ。今までは DSM というと activity の順序性を表現することのみを指していた用語であったが、DSM（Design Structure Matix）、DMM(Domain Mapping Mtrix)、MDM(Multiple-Domain Matrix)という３つのカテゴリでマトリックスの種類を整理している（http://www.dsmweb.org/）。その整理学では DSM（Design Structure Matix）は基本となるもので１つのドメイン内の要素関係を整理することが可能である。例えばタスク間の関係を表現するために利用される。また DMM(Domain Mapping Mtrix)は２つの異なったビュー（またはドメイン）の関係性を整理するために用いられるもので、タスクと作業者の関係など２つの領域に跨った要素関係が表現される。そしてまた、MDM(Multiple-Domain Matrix)は DSM と DMM を結び付けてひとつの完成したモデルとして表記する方法である。つまり複数のドメインに跨って要素間の関係を整理しつつ、単一のドメイン内の要素関係についても整理を行うマトリックス表現方法である。このように今までは単純に activity を要素としたマトリックスとして DSM の考え方が、マトリックスの要素に関わりなく個々に自由に要素を当てはめてマトリックスを利用することが広まってきたために、広義性を増し３つの用語（DSM、DMM、MDM）で整理されるようになった。

＜Axiomatic Design について＞

また先行研究の動向として別の観点からは、製品を構成する依存関係について研究を進めているのが東大の藤本教授であり、モジュラー型と擦り合わせ型開発というキーワードで様々な論文を発表されている。藤本教授は機能構造マトリックス（function-structure matrix）ということで製品の構成要素の依存関係に注目されており、その製品アーキテクチャの重要性について述べられている。その基礎理論となっている研究が MIT Nam Pyo Suh（現 KAIST）による Axiomatic Design（公理的設計）であり、設計理論について述べられたものである。

顧客領域、機能領域、実体領域、プロセス領域に分けて、設計において守るべき原則を公理（反例や例外がない自明な心理や根本的な事実。公理は他の自然法則や原理から導けない（Nam Pyo Suh(2004)から引用））を元に導いている。その解説では、各４領域が互いに写像関係にあることを利用してマトリックス（行列）でその関係性を表現し解説を試みている。特に公理１：独立公理、公理２：情報公理の２つを基本とした設計理論である。独立公理とは、最小の独立した機能で設計されることが望ましい、という公理である、つまり、列に“機能”、行に“要求条件”として行列表現した場合には対角行列の関係にあることが最善の設計である、という公理である。次に情報公理とは、独立公理を満たす設計の中で最良な設計とは複数の設計情報で設計されるのではなく、単一の情報により機能設計を進めていけるものである、という公理である。

比較的、公理的設計の考え方は全般を捉えて議論されているが、成果物やチームという要素については言及されることなく終わっている。また、機能とそれを実装している部品を同一視して扱っている傾向がみられる点は社会システム、情報システムなどへの応用度に難があると考える。

＜MDM について＞

このようにマトリックスを利用した研究は現在様々な観点で進められており、そして最近の整理学上では DSM、DMM、MDM のどれかには位置づけられるものではあるが、どれも一長一短なところがあり、製品開発に関わるオペレーション全てを上手く捉えきれたものではないことはこのように明白である。これは MDM でも同様な様子であり、オペレーション全体を捉えて複数のドメインの関係性を整理しようとはしているが、現在のところその要素として何が的確なものであるか有効な提案はなく、MDM という考え方のみが先行している様子である。Technical University Minchin　Udo Lindmann, Maik Maurer, Thomas Broun（2009）では MDM を用いたマネジメント手法を紹介しているが、定性的な議論に留まり定量的な評価までは行えていない。また、紹介されている MDM は、本研究のマトリックスとは異なり、すべてのドメインを同一のマトリックス上に表記して整理している。行と列をそれぞれ、ある領域は例えば function で別の領域は activity の領域というように記述し、すべてを混在させている。このような表記の場合は、非常に巨大なマトリックスになるため全容が分かりにくく、またそのマトリックスの作成も専用ソフトなしには容易に行うことができないであろう。

＜本研究の位置付け、新規性＞

このような状況を踏まえ、本研究では公理的設計の考え方を発展させ機械工学系からソフトウェア開発の世界へ導入を試みているものと捉えることができる。その中で特に新規性が高い点は、

- ✧　元々の公理的設計の対象よりも広く（needs~function~team）対象を捉えたこと

✧　Cost、Quality をマトリックスの性質から算出する方法を導き出したこと

更に言いかえるならば、ソフトウェア開発において各種技術導入による生産性向上効果を理論計算可能にしたことなど、旧来のソフトウェア工学では数個の事例を列挙するのみしか証明できなかったことを、本研究で提案したモデルによって、その裏付けとなる理論をもとに論証することを可能にしたことである。

①　プロダクト品質から組織活動品質へ（旧来のシステムエンジニアリングと比較して）

本研究で提案したいことの 1 つ目として、このようにある製品やサービスを生み出す仕組み全体を捉えて、“品質“と定義することを提案する。今までの品質と言えば、組織活動の結果として出来上がったアウトプットに対する品質を評価するのみであった。しかし、その評価方法もあらゆる観点で評価・検証しているわけではなく、当然そのテスト観点には漏れがある。極論すればテストにパスしたのは偶然なのかもしれない。前述の通り、その製品を作るまでには、”人（組織)”、“作業プロセス“、そのプロセス途中で作られる”成果物“という要素があり、それぞれが上手く噛み合ってこそ高品質なものが出来上がるのである。今までのように途中成果物と最終成果物でしか評価・検証しない時代はもう終わったのではないか。ソフトウェアに関わらず、食品業界で言うなら”毒入り餃子“、金融業界で言うなら”サブプライムローン“など一見良さそうに見えても実際は非常に低品質な商品も多い。商品という最終成果物だけでなく、それを作る組織活動そのものの評価・検証も非常に重要なことであると考える。そして、組織活動そのものを評価することで、出来上がってきたものを評価し改善するというフィードバック・ループ(Feedback- loop)またはリアクティブ（Reactive）な改善ではなく、組織活動評価を元にフィードフォワード（Feed forward-loop)、プロアクティブ（Proactive）に対処していくことで最終的な成果物の品質も高まるのである。

よって、この公理的設計を拡張したモデルをソフトウェア開発プロジェクトに適用することで実現したいことは、「To-By-Using」としてまとめると以下のようになる。

- Using：情報システム開発プロジェクトに対してマトリックス分析を用いることによって見える化し、
- By：そのハブとなる要素やボトルネックを明確化して情報の流れを円滑にすることによって、
- To　：失敗リスクの軽減、品質や生産性を向上させる。

ということになる。

②　本研究で考えたモデルの特徴

ソフトウェア開発においては成果物、例えば設計書などを含むドキュメントやソースコードなどが成果物として重要な要素であり、またそれを作成する技術者チームを無視することはできないため、本研究では考慮に入れた。また Nam Pyo Suh 教授（2001）の述べている公理的設計では写像関係を基本概念として入れているが、本研究の基本概念は開発組織活動（実質的には開発如何に関わらず、企業活動などの様々な組織活動全般に通用すると考えるが）を構成する要素群（オブジェクトと考えてもよい）の関係性をネットワークとして考え、そのネットワークを伝播する情報がどこかで滞るために開発プロジェクトに問題が発生すると考え、如何に円滑にそのネットワーク上で情報が各要素間で伝播していくようにできるか、という観点で考えたモデルである。現代の大規模で複雑なシステムでは、システムを構成する要素そのものの難易度も上り、また要素間を複雑にネットワークがつながって情報を伝播していると捉えることができる。そのネットワーク的に捉えた各

要素の関係性を行列（マトリックス）で整理し、それを行列計算により情報のボトルネックを評価分析し、最適解を導く手法をまとめることとした。

よって、対角行列が干渉もなく一番スムーズに情報が伝達されることから、対角行列（単位行列）を基準にして議論を進めていく。

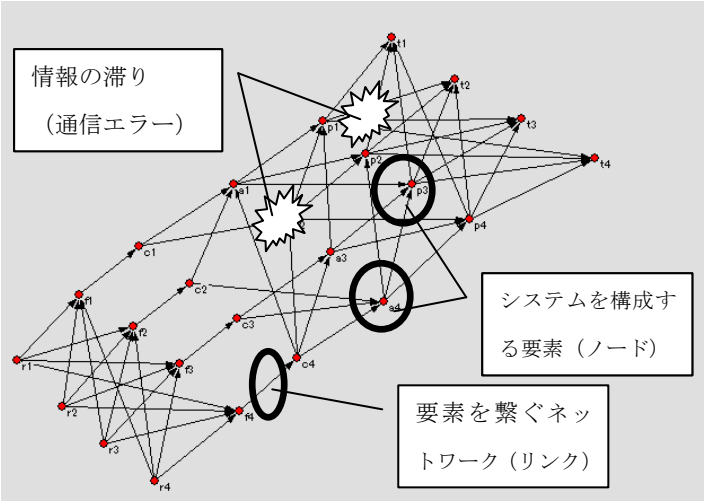


図 32　システム構成のネットワーク的表現

システムの品質を本章のはじめで以下のように明記したが、ネットワーク的な観点からその方程式を書き換えると以下のような等式が成り立つ。

<u>情報システムの品質</u>

＝“システムを構成する要素（利用する技術、利用する入力情報（設計書など））の品質”

×“作業プロセス（作り方）の品質”

×“作業者の能力（技術力・コミュニケーション力 etc）”

＝“構成要素の難易度”×“ネットワークの複雑性”

と考えることもできる。例えば企業をひとつのシステムに捉えたとき、それを構成する人材に対しては様々な研修機会などによりその育成（改善）に努めている。しかし、その構成要素間のつながりを簡素化する方策がどれほどあるのであろうか。現在ではインターネットの発達により、人とのつながりも非常に複雑化しており、人材の能力は向上してもネットワークの複雑性が増しており、結果として能力向上分が帳消しになっているとも考えられる。小規模のシステムでは多くの場合、ネットワークの複雑性は微量であり、人材の能力向上による効果の方が効くことが多く、小規模なシステムでその難易度について問題になるケースは少ない。

そのように、システムを単に俯瞰的に捉えるだけでなく、その構成要素全体のつながりを分析することは有益な効果が見込める着眼であると考えた。

先行研究で行われているマトリックスの利用方法では、ソフトウェア開発業務を例にした場合に以下のような不具合がある。たとえば、ソフトウェアの設計では要求仕様から必要な機能を洗い出し、その機能を実際にどの部品に実装するか設計しなければならない。その設計書を作成するための作業プロセスもあり、当然そのプロセスに合わせて動く開発チームが存在するのである。作業プロセスのみに注目した DSM では、プロセス

を改善してもチーム構成については言及されない。そのようにプロジェクト内部に存在する構成要素を鳥瞰的に把握する仕組みが存在してなかったのである。

このようにマトリックスを利用した先行事例はあるが、先ほど述べたようにプロジェクトの全容を現したものではないことが分かる。よって、今回プロジェクトを構成する要素の関係性をマトリクスで整理し、プロジェクト全体のモデル化を行うことを提案する。以降では具体的な行列式に落としたモデルを用いて説明を行うが、本例では極力シンプルなモデルとするため、以下の５つのマトリックス関係に絞って説明を行う。

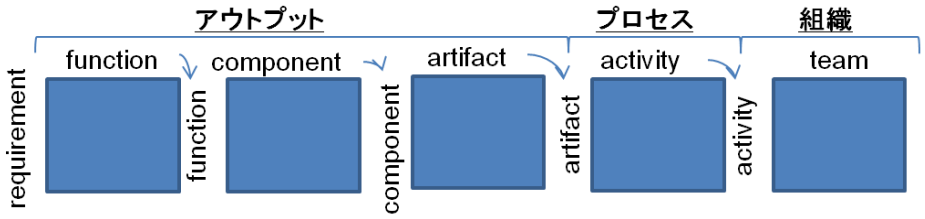


図 33　説明用サンプルモデル

③　コンセプト説明（情報を円滑に伝える仕組みとは）

これを簡単な行列を用いて更に議論を続ける。

＜コスト＞

	f1	f2	f3	f4
r1	1	0	0	0
r2	0	1	0	0
r3	0	0	1	0
r4	0	0	0	1

左図は“要求仕様”と“機能”の関係性を表した行列である。この行列では要求仕様と機能が１：１の関係になっている。(r*は要求仕様項目（requirement）を示し、f*はそれを実現する各機能（function）を示す。)

	c1	c2	c3	c4
f1	1	0	0	0
f2	0	1	0	0
f3	0	0	1	0
f4	0	0	0	1

左図は“機能”とその機能を実装する“部品（コンポーネント）”との関係性を表した行列である。これも機能と部品を１：１の関係にしている。(c*は各部品（component）を示す。)

	a1	a2	a3	a4
c1	1	0	0	0
c2	0	1	0	0
c3	0	0	1	0
c4	0	0	0	1

左図は“部品（コンポーネント）”と“成果物（設計書など）”の関係性を示した。これも同様に１：１の関係である。(a*は artifact=成果物を示す。)

	ac1	ac2	ac3	ac4
a1	1	0	0	0
a2	0	1	0	0
a3	0	0	1	0
a4	0	0	0	1

左図は“成果物（設計書など）”とその作成作業を意味する“activit”の関係性を１：１で示した行列である。(act*は各プロセス（activity）を示す。)

	t1	t2	t3	t4
ac1	1	0	0	0
ac2	0	1	0	0
ac3	0	0	1	0
ac4	0	0	0	1

左図は“activity”と作業スを実行する“チーム“の関係性を１：１で示した行列である。(t*はチーム（team）を示す。)

図 34　サンプルモデル（１）の関係行列

これらの関係は要求仕様からそれを実現する機能、部品、またそれを作る人まで全て１：１に割り当ててあるということを意味する。これをイメージし易いように視覚化するため、ネットワーク図で表現する。本図の作成にはネットワーク描画ソフト（フリーソフト）“Pajek”を利用した。(http://vlado.fmf.uni-lj.si/pub/networks/pajek/よりダウンロード)

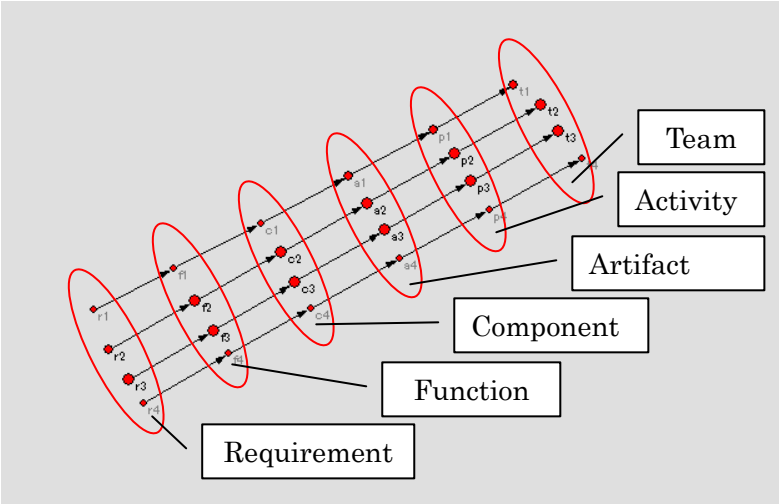


図 35　各要素のネットワーク関連図（1）

先ほどの繰り返しになるが、各要素が１：１の関係であり、ひとつのチームが単純にひとつの要求項目を実現する機能を作るケースである。このようなケースでは非常にプロジェクト全体も可視化し易いことは容易に判断できるであろう。各担当が相互に依存関係がなく作業ができることはマネジメントもし易く、開発するサービス・製品の品質にも良い影響がある。例えば、レストランで４人のお客様からオーダーがあった場合、それぞれを４人のシェフが作ることを想像してもらえばよい。出来上がりは各シェフの力量、またその作業のやり方次第なのである。しかし、最初に述べたとおりこのモデルでは基本要件（feature）やニーズ（needs）の関係を省略している。

行列式で表したモデルであるので、モデルの評価を数学的に行うことが可能となっている。上記例を行列計算で処理を試行してみる。その場合、要求仕様（requirement＝r1,r2,r3,r4）とチーム（team＝t1,t2,t3,t4）の関係性は以下のようになる。

$$\begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \underset{\text{Requirement-Function}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \underset{\text{Function-Component}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \underset{\text{Component-Artifact}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \underset{\text{Artifact-Activity}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \underset{\text{Activity-Team}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix}$$

となり、r=t の関係が成立していることが分かる。これは前述のネットワーク図で示した通り、すべてが１：１で関係付けられていることを数学的に表現し直したものである。また、この行列計算式はソフトウェア開発において、各チームの工数が分かれば各要求仕様の開発にそれぞれどの程度の工数がかかっているか算出することができることを示しており、つまり工数＝開発費とも置き換えられるので、各要求仕様の原価が理論計算可能となることを示している。これまでのようにソフトウェアプログラムのライン数からの算出ではなく、その開発原価の算出が可能になった。

例えば、前述のケースでは、チーム１（t1）に対して 100 万円かかっていたならば、要求仕様１（r1）の実現には 100 万円の開発原価がかかっているということを示している。

そしてまた、この行列式の逆行列を用いれば、（要求仕様単位の売上額まで分かっていればという前提は付くが）各チーム単位の売上高を算出することが可能になる。つまり、上記の開発原価と逆行列式から算出される売上高から、各チームの利益率の算出も可能となる。

$$\begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \times \begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix}$$

次に品質面について考えてみる。本事例の場合ではチーム（t）をその能力と考えれば、要求仕様もその能力に応じたものになる。具体的にはチームの能力を 10 段階評価するとした場合、t1=10、t2=7、t3=5、t4=8 と設定する。

$$\begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 10 \\ 7 \\ 5 \\ 8 \end{pmatrix} = \begin{pmatrix} 10 \\ 7 \\ 5 \\ 8 \end{pmatrix}$$

このように、r1=10、r2=7・・・というようにチーム（t）の能力がそのまま反映されることとなる。

このように数値的な評価を行っておくことで、実際の開発においてもどの要求仕様が品質的に問題になりそうか予め予測することができ、試験工程なでのチェックは勿論、設計工程でのチェックも、そのポイントが絞られているためプロアクティブで効率的な品質管理が可能となる。

品質に関する応用方法としては、作業の複雑性などを加味するが可能である（複雑性の考慮方法は後述）。プロセスと成果物の行列式において、その数値的考慮を入れることにより、品質算出が可能となる。例では、通常“１”の精度で可能な作業のひとつを“０．７“の精度で実施した場合をみてみよう。

$$\begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \underset{\text{Requirement-Function}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \underset{\text{Function-Component}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \underset{\text{Component-Artifact}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \underset{\text{Artifact-Activity}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \underset{\text{Activity-Team}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix}$$

このような行列計算で表現することが可能となる。先ほどの例（チームの能力を 10 段階評価するとした場

合、t1=10、t2=7、t3=5、t4=8 と設定）を用いて、実際に計算を行ってみる。結果、r1=10、r2=4.9、r3=5、r4=8 となり、r2 の品質期待値が最低になってしまうことが分かる。

$$\begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 10 \\ 7 \\ 5 \\ 8 \end{pmatrix} = \begin{pmatrix} 10 \\ 4.9 \\ 5 \\ 8 \end{pmatrix}$$

このサンプルでは省略しているが、ニーズ、基本要件などを考慮して、この品質で問題がないならば構わないが、要求仕様 2（r2）に対して最低の品質期待値が“７”以上を求められるときは、逆行列式を用いてチーム 2（t2）の要素を算出すれば良い。

$$\begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \times \begin{pmatrix} 10 \\ 7 \\ 5 \\ 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 10 \\ 10 \\ 5 \\ 8 \end{pmatrix}$$

結果、チーム 2（t2）の要素としては“１０”の能力レベルが必要ということが示された。チーム編成の再考などが必要ということになる。このようにチーム編成を変えた場合の開発費への影響も当然考慮すべきで、品質と開発費の最適解を見出すことが要求される。

次にもう少し複雑な例を用いて説明を補足する。以下のように n:n で関連性をもったケースを考える。

1．要求-機能＝D	r=D･f			
	f1	f2	f3	f4
r1	1	1	1	1
r2	1	1	1	1
r3	1	1	1	1
r4	1	1	1	1

2．機能-部品＝A	f=A･c			
	c1	c2	c3	c4
f1	1	1	1	1
f2	1	1	1	1
f3	1	1	1	1
f4	1	1	1	1

3．部品-成果物＝B	c=B･a			
	a1	a2	a3	a4
c1	1	1	1	1
c2	1	1	1	1
c3	1	1	1	1
c4	1	1	1	1

4．成果物-作業＝C	a=C･p			
	ac1	ac2	ac3	ac4
a1	1	1	1	1
a2	1	1	1	1
a3	1	1	1	1
a4	1	1	1	1

5．作業-チーム＝E	p=E･t			
	t1	t2	t3	t4
ac1	1	1	1	1
ac2	1	1	1	1
ac3	1	1	1	1
ac4	1	1	1	1

図 36　サンプルモデル（２）の関係行列

この関係を行列式で表すと以下のようになる。

$$\begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \underset{\substack{\text{Requirement-} \\ \text{Function}}}{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}} \times \underset{\substack{\text{Function-} \\ \text{Component}}}{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}} \times \underset{\substack{\text{Component-} \\ \text{Artifact}}}{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}} \times \underset{\substack{\text{Artifact-} \\ \text{Activity}}}{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}} \times \underset{\substack{\text{Activity-} \\ \text{Team}}}{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix}$$

$$= \begin{pmatrix} 256 & 256 & 256 & 256 \\ 256 & 256 & 256 & 256 \\ 256 & 256 & 256 & 256 \\ 256 & 256 & 256 & 256 \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix} 42$$

サンプルモデル（1）のように対角線の成分のみ“1”で他の成分が全て“0”であるような単位行列ではない。つまり、各成分が全て関連付いていることを示し、下図のような複雑な関係性を持っている。このように全ての要素が絡み合っている場合、末端にあるチーム間での情報共有が混乱し易くなるのは想像するのも容易であろう。また、プライムコンストラクターがこの要素の中の一部を行い、その他を請負企業に任せてしまうような現在の開発のやり方では“どのように実際に設計、実装されているのか全く分からない”ということもうなずける。

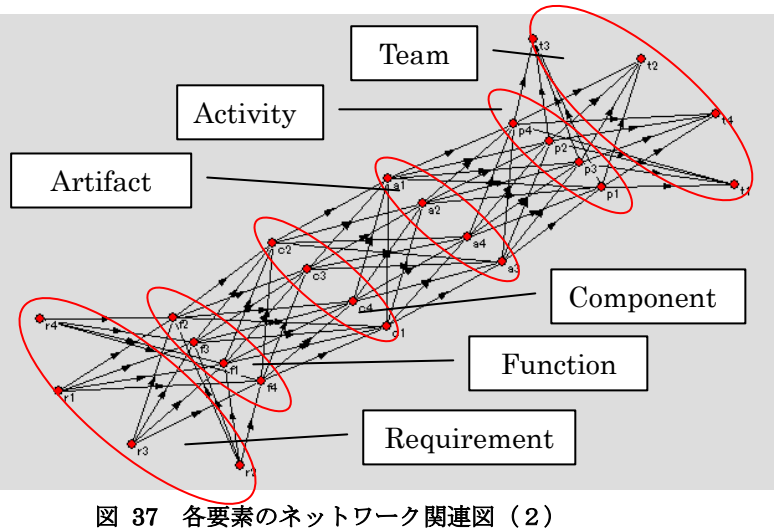


図 37　各要素のネットワーク関連図（2）

このような状況を如何に変革していくか、そのソリューションを本研究でも考えていくこととする。

複雑な要素間の関連性をもっている場合でも、行列式によりモデル化したことによりサンプルモデル（1）のように原価計算なども可能である。各成分が“256”になっているが行列による変換で原価金額が大きくなることは有り得ないので、ノルムにより割った行列を用いて説明する。

$$\begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \begin{pmatrix} 256 & 256 & 256 & 256 \\ 256 & 256 & 256 & 256 \\ 256 & 256 & 256 & 256 \\ 256 & 256 & 256 & 256 \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix} \Rightarrow \begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix} \times \begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix}$$

このように各チーム（t）の工数に要した開発費用の 0.25 倍ずつが各要求仕様（r）の原価として足し込まれていることが分かる。例えば要求仕様 1（r1）の原価を考えてみると、各チームにかかる費用が 100 万円ならば 25 万円ずつの開発費が各チームから足し込まれ、合計として 100 万円が要求仕様 1（r1）の原価であると言える。また各チームに要した開発費が（t1,t2,t3,t4）＝（100 万,80 万,60 万,40 万）とした場合は、以下のように各要求仕様ともに 70 万円の開発原価と算出される。

$$\begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix} \times \begin{pmatrix} 100 \\ 80 \\ 60 \\ 40 \end{pmatrix} = \begin{pmatrix} 70 \\ 70 \\ 70 \\ 70 \end{pmatrix}$$

また各要求仕様に対する各チームの売上高を算出することも可能である。しかし、この事例の場合は行列式

=0 であり逆行列を算出できないため、擬似逆行列を求めてそれにより計算を行っていく。なお、擬似逆行列算出にあたってはフリーソフト“SciLab”（http://www.scilab.org/）を用い、その計算プログラムは（http://staff.aist.go.jp/toru-nakata/Gauss/Gauss2.html）を元に編集して利用した（参考資料として添付）。

$$\begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix} = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}^{+} \times \begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix} = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix} \times \begin{pmatrix} r1 \\ r2 \\ r3 \\ r4 \end{pmatrix}$$

となり、各要求仕様（r1,r2,r3,r4）=（100 万円,90 万円,80 万円,70 万円）の売値とした場合、

$$\begin{pmatrix} t1 \\ t2 \\ t3 \\ t4 \end{pmatrix} = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix} \times \begin{pmatrix} 100 \\ 90 \\ 80 \\ 70 \end{pmatrix} = \begin{pmatrix} 85 \\ 85 \\ 85 \\ 85 \end{pmatrix}$$

各チームの売上は 85 万円と算出される。合計は 85 万円×４チーム=340 万円。先ほど計算した原価の合計 280 万円から考えると利益は 60 万円になるが、各チームの利益を比較すると

チーム原価：(t1,t2,t3,t4）=（100 万,80 万,60 万,40 万）

チーム売上高：(t1,t2,t3,t4）=（85 万,85 万 85 万,85 万）

であるから、利益は（t1,t2,t3,t4）=（-15 万,5 万,25 万,45 万）となり、チーム１（t1）の生産性の悪さが確認され、逆にチーム４（t4）の生産性寄与度が高いことが分かった。このように行列式により開発費用に関する分析にも役立つことが分かる。

以上の事例により、大枠の考え方は理解して頂けたと考えるが、コスト（Quality）・品質（Cost）・開発期間（Delivery）の本モデルでの考え方について整理しておく。

コストについては前述の通りであるが、以下のとおり team を工数と単位とした値にするのか、具体的な原価ベースで計算するのか、によって行列成分の値の取り方が異なることに注意が必要である。

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

この行列計算を下記のように置き換えると

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

となり、ベクトル r に対する各チームの原価算出を行う場合は、

$$\vec{t} = A^{-1}\vec{r}$$

正規化するために行列式=１であることが必要である。１以外の場合は行列により、ベクトルの大きさが増大してしまうことになり、正確な原価にならない。逆に、ベクトル t の工数を元にして売値を付けるときには、各行列の成分の値は付加価値分を含むこととなり、行列式が１である必要はない。

難易度の設定方法、つまり各行列成分の値の決め方は、それぞれの依存関係をまずは整理すること。次にそ

の依存度を評価し、その比率により決定すればよい。実際に利用するに当たっては、コストを評価するフェーズに応じた依存度の評価が必要で、例えば開発作業を開始した当初に行う評価では、ある程度の粗さは許容すべきである。プロジェクトマネジメントを行う立場のグループでディスカッションしながら仮置きとしてでも数値を決定することで、今後プロジェクトを遂行していく上での様々なシミュレーションを行うことができ、数値の精度を競うよりも、そのシミュレーションによって何パターンかの計算を行い、そこから得られる将来発生し得る事象への対応力をつけることの方が有益である（但し、正確な見積もりが可能ならばそれに越したことはない、ということは当然のことである）。また逆に開発作業後半での開発費の積算においては、例えば今までのどの作業にどのチームがどの程度の稼働状況にあるのか、などを正確に把握することで行列成分の数値を決定することは重要である。また逆に、開発後半になっても行列の成分値を正確に把握できないような状況であるということは、プロジェクトの作業状況、プロジェクトの作業の仕組みをしっかりと管理できていないということと同意であり、そのプロジェクトの作業品質に問題があるということにもなる。

＜クオリティー＞

品質は、前述の繰り返しとなるが

システムの品質＝
“システムを構成する要素（利用する技術、利用する入力情報）の品質”
×“作業プロセス（作り方）の品質”
×“作業者の能力（技術力・コミュニケーション力 etc）”

で表すことができる。これをマトリックス分析では、

システムを構成する要素=function-component、component-artifact、
作業プロセス=artifact-activity
作業者の能力=team-activity

以上のマトリックスで表現することができる。よって、個々のマトリックスの品質を評価することで、分析対象としているシステム全体の品質を評価することが可能なのである。そこで、マトリックスの品質はどのように評価できるか考えると、

品質＝複雑性×難易度　（複雑性＝リンクの数、難易度＝ノード自体の持つ難易度）

と定義できる。

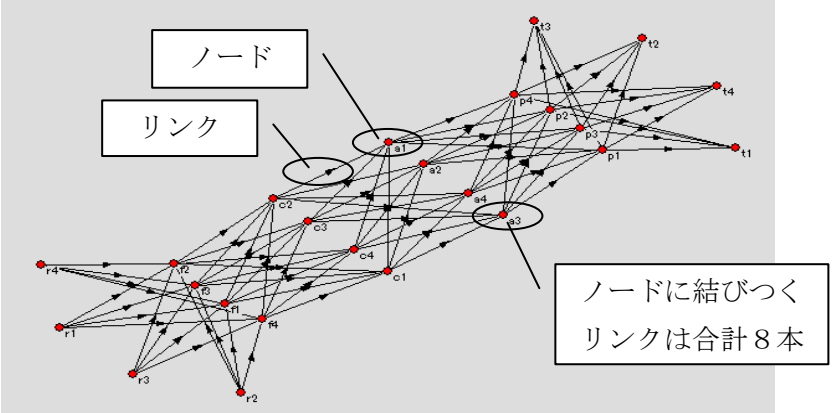


図 38　ノードとリンクとは

これをネットワークの関係性を行列に置き換えて数値化することにより、定量的に算出する方法を述べていくが、その前にノードとリンクについての説明を追記しておく。ノードとは行列の各成分に該当する component や function などを指し、そのノード間を繋げるものをリンクと呼んでいる（上図参照）。前述のサンプルモデル（１）ではひとつのノードに対してリンク数は２つ（末端の”要求仕様“と”チーム“は１つ）だけであるが、サンプルモデル（２）ではひとつのノードが８つのリンクを持っている（末端の”要求仕様“と”チーム“は４つ）ことになる。

ここから品質を定量的に算出する方法についての説明を始めるが、まず前提として単位行列（対角成分のみが１で他の行列成分は０）がすべての基準であるとする。これは、Nam Pyo Suh（2001）でも述べられている公理であり、つまり、独立公理と呼ばれるもので、例えばあるひとつの機能は複数の部品で実現されるよりも、ひとつの機能はひとつの部品で実現する必要がある、というものである。それは、複数の部品で構成する場合、互いの部品性能が影響しあうことにより、互いに独立関係にある場合に比べて、本来実現すべき機能のロバスト性（外部・内部の変動に強い性質、丈夫さ）が損なわれるからである。

そのときに、複雑性は単位行列からの歪み度で定義する。単位行列はリンク数が１つのノードに対して、出と入りの１つずつしか定義されない。しかし、その他の行列ではリンク数が増加し、ノード間の関係性を複雑化している。リンク数が増加すれば必然的に行列の同一行内には 0 以外の値が埋まることになるので、リンクの複雑性は行列が対角行列なのか三角行列なのか全行列なのか、など行列の形態をみることで判別が可能になる。行列の対角成分以外にどの成分が 0 以外の値を持っているのかを分析することで歪み度を定量化することができる。これを数学的に分析して定量化するには、**システムの関係性を定義した行列の固有ベクトルと単位行列の単位ベクトルの内積の値を難易度として定義**し、評価すればよい。この考え方は、情報検索用アルゴリズムのひとつであるベクトル空間モデルとも非常に似た考え方であり、ベクトル空間モデルでも内積による $\cos\theta$ の値でその類似性及び相違性について論じている。

また、難易度については個々の構成要素の難易度は行列成分の値で示すこととすれば、行列全体の難易度つまりシステム全体の難易度は、**ノルム（最大特異値）の逆数が難易度**と定義できる。つまり行列によりベクトルの大きさにどの程度変化が生じているかを評価することで難易度を分析する。但し、難易度が上がるほどノルムが大きくなると品質値もそれに応じて大きくなることは複雑性と矛盾するため（$\cos\theta$ は θ <90 度までは減少関数）、ノルムの逆数を難易度とした。

最後に、各行列成分値の設定方法であるが、単位行列を基準としているので値=1 が基準値となる。難易度が上がる場合は 1 以下の値を示し、0 に近づくほど難易度が高いことを示している。実際に難易度を設定するときには、例えば、他に事例の無い未経験の技術を導入する場合=0.2、他に事例はあるが自チームに経験者がいない場合=0.4、自チームに経験者が数名いる場合=0.7 などというように、独自の指標値を決めてからチーム内で議論して具体的な行列成分値を設定することが望ましい。

では、以降で単純な 2×2 の行列で品質値を算出する具体例を考えてみる。

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

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

次に複雑性を算出するため、まず固有値・固有ベクトルを求める。

$$\begin{pmatrix} 1/2 & 1 \\ 1 & 1/2 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} 1/2-\lambda & 1 \\ 1 & 1/2-\lambda \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

これにより、固有値λを算出すると、

λ=3/2、-1/2

となる。よって、ノルムを求めると

最大特異値=｜3/2｜=3/2

となり、難易度は逆数をとり 2/3 と算出される。

また固有ベクトルを計算すると

λ=3/2 ならば固有ベクトル a1=（1,1）

λ=-1/2 ならば固有ベクトル a2=（-1,1）

よって、A=3/2×a1＋（-1/2）×a2=（2,1）

$$\vec{A}\cdot\vec{e} = (2,1)\cdot(1,1) \quad = 2\times1+1\times1 = 3$$

$$\vec{A}\cdot\vec{e} = |\vec{A}||\vec{e}|\cos\theta$$

$$\cos\theta = 3/\sqrt{10} \fallingdotseq 0.949$$

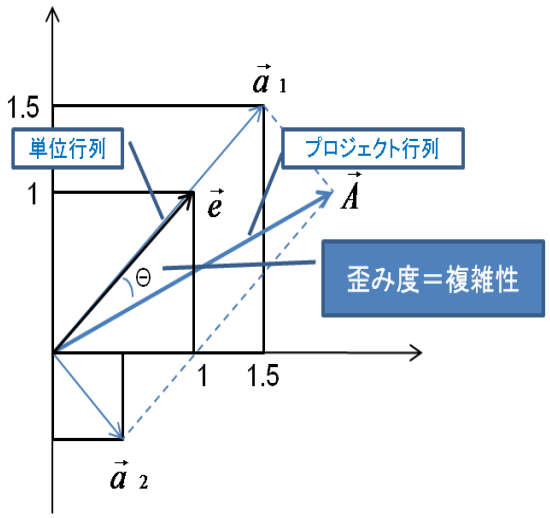


図 39　複雑性のベクトル空間的意味

よって、複雑性は 0.95 となる。

以上により、システム全体の品質は複雑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、

2/3×0.949=0.632≒<u>0.63</u>

となる。

＜品質を考える上での行列計算のパターン分析＞

以降では、行列計算の特性からみた、システムのロバスト性を考えてみる。各関係性を示す行列の複雑性のパターンは 7 種類考えてみる。①は needs や requirement、そして team などが対角化行列、つまり互いの要素が独立性高く出来上がっているケースであり、その間のプロセス（アクティビティ）などの行列は全行列で

あり、複雑な関係性を表現している。②はその逆パターンであり、③は顧客・市場に近いところではシンプルな関係性（対角行列）であるが、次第に三角行列、全行列と複雑度が増すパターン、このように各種のパターンについて、その行列が表現するシステム全体の複雑性について整理してみる。７つの行列すべての組合せを考えても分かり難いだけであると考えて、３つの行列の関係で整理することとした。対角行列（対角成分のみが 0 以外であり、それ以外はすべて 0 を成分とする行列）、三角行列（行列の上三角形または下三角形が全て 0 の成分である行列式）、全行列（全要素が 0 以外の数値である行列）を以下のような表記に置き換えて、その特性を述べる。

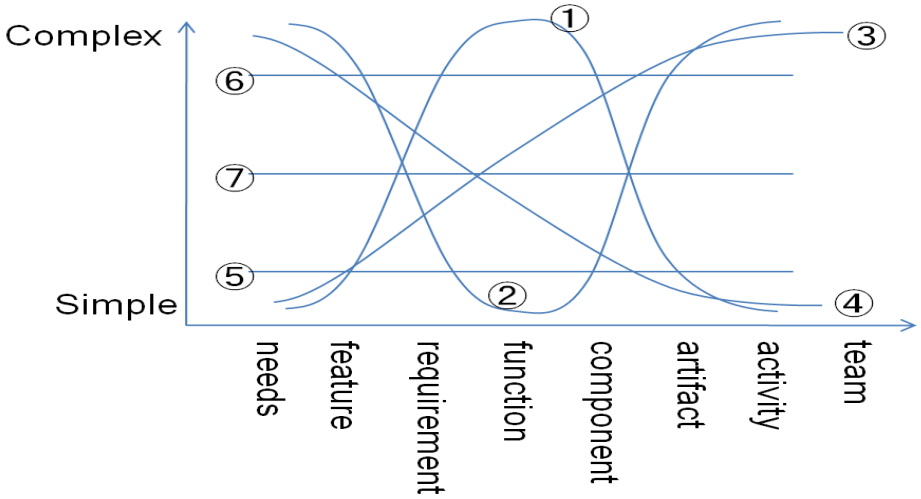


図 40　各行列の複雑性とシステム全体の複雑性について

対角行列で挟まれた全行列は全行列になる①、また対角行列×三角行列×全行列も全行列③、三角行列×三角行列×三角行列は三角行列（但し３つ共に上三角形または下三角形であること）⑦というような性質がある。ロバスト性という面では⑤が一番強いシステムであり、次に⑦、③’、④’が強いシステムであると言えそうである。行列成分の値まで考慮していないため、非常に粗い見方ではあるが、おおよその性質としてこのようなことが言える。正確に分析するには、システムを構成する行列が全行列でもその次元数（行・列の数）が小さいならば、次元数の多い三角行列よりも複雑性は低い。また行列成分の値（＝難易度）によっても、そのシステム全体品質は異なることは留意しておくべきである。このような留意点はあるにしても、下記の計算パターンは大規模且つ複雑なシステムをコントロールするためのヒントが潜んでいる。

$$[\backslash][X][\backslash] = [X] \rightarrow ①$$
$$[X][\backslash][X] = [X] \rightarrow ②$$
$$[\backslash][LT][X] = [LT][X] = [X] \rightarrow ③$$
$$[X][LT][\backslash] = [X][LT] = [X] \rightarrow ④$$
$$[\backslash][\backslash][\backslash] = [\backslash] \rightarrow ⑤$$
$$[X][X][X] = [X] \rightarrow ⑥$$
$$[LT][LT][LT] = [LT] \rightarrow ⑦$$
$$[\backslash][LT][LT] = [LT] \rightarrow ③'$$
$$[LT][LT][\backslash] = [LT] \rightarrow ④'$$

$[\backslash]$=対角行列
$[LT]$=三角行列
$[X]$=全行列

（Nam Pyo Suh (2001) p.44 を元に編集）

図 41　行列計算によって導かれる性質

現在の大規模な不雑なシステムでは、一足飛びに⑤のタイプにすることは非現実的な話であるが、地道な改善により⑦、③‘、④’のパターンにまでシステムを修正することもアプローチとしては良い。例えば artifact 自体を分離または統合して要素の独立性を高くすることも対策としては考えられる。また別の考え方としては artifact を管理するドキュメントを作ることにより、artifact-artifact の関係性を示す行列を 7 つの行列関係の間に挿入する。Artifact 同士の関係性を示した行列が固有値・固有ベクトルから導かれた対角行列であれば、元の行列は対角化される。または行と列の入れ替えを適切に行うことで対角化または三角行列に変化させる方法もある。

その他の考え方としては、例えば①のパターンでは両端の行列は対角行列であり、その両端の行列関係については行列成分で示される難易度を如何に下げるか、という観点のみに注力すればよい。そして、問題となる間に挟まれた全行列を注意深くその関係性について注意することにポイントを絞ればシステム全体の動きをコントロールすることが可能であると考えられる。例えば、前述したとおり、現在の情報システム開発はゼネコン的な体制で開発がおこなわれており、多くの作業を協力会社に請負で出している。その請負先企業のシステムが全行列を含んだ仕組で仕事がなされている場合、発注側は複雑な関係性のなかでやり取りされる情報を知ることなく作業が進んでしまうことになる。このような場合、全行列に近いところこそ情報のハブとして、そこでやり取りされる情報を把握しておくことが大規模複雑なシステムをコントロールするためのキーである。

情報把握を怠ったような状況では、請負企業からの納品物を検査するだけで十分な品質が担保されることはない。それは現状の様々なトラブルが証明している事実である。しかし、複雑な行列部分までを自社で作業を行い、それ以外の対角行列または三角行列部分を請負企業に仕事を任せることは正しい選択である。何故なら、ロバスト性の高い仕組で作業がなされているので、その請負企業で起こる様々な変化に対しても問題なく処理され、一定の品質を保てるからである。このような考え方は現在のゼネコン体制による開発を改善していく上でのヒントが隠されているように考える。よって後述の中でその具体案を提案していく。

＜デリバリー＞

次に開発期間（デリバリー）についての説明を行う。例えば、下図に示すように作業 P1 によって成果物 a1 が作られ、その a1 をインプットとして作業 P2,P3 が行われる。その二つの作業の結果が成果物 a2 に反映され、次の作業 P4,P5 のインプットになる。最終的には P4,P5 の作業の結果として成果物 a3,a4 が出来上がって、一連の作業を終了する。という流れを考えたときに、その作業期間はどのように考えればよいか説明を行う。

まず考えるマトリックスは team-activity と activity-artifact の２つである。デリバリーに依存するのはプロセスであるのでこの２つのマトリックスを考えればよい。そして先ほどの作業フローをアクティビティ p と成果物 a のマトリックスにして整理する。アクティビティ p に対してどの成果物 a が output または input になっているかを書き込み、次にそのマトリックスの中で output となっている成分のみ抽出して行列として抽出する。また同時にチーム t とアクティビティ p の関係を整理する。どのチームがどのアクティビティを行うのか、マトリックスで整理するのである。下図の例ではチームがそれぞれ別のアクティビティを担当しており、互いに独立関係にあるものとした。そのような前提で、各 team の作業日数を(t1,t2,t3,t4,t5)=(10,20,10,5,10)とすると成果物作成に要する日数は（a1,a2,a3,a4）=（10,30,5,10）と算出される。a3 と a4 は並列作業で作

成されるものであるので、長い期間がかかる a4 が終わって全作業が完了することになる。よって、全体に要する作業期間は a1+a2+a4=10+30+10=50 日となる。

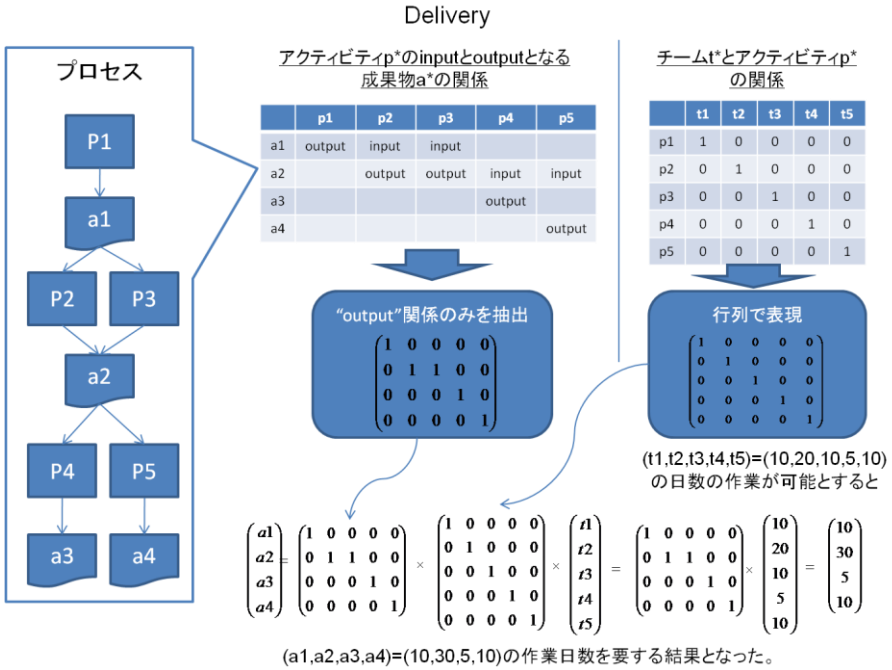


図 42　Delivery 計算のサンプル（1）

次のサンプルは作業フローが異なり、分岐と結合を含んだ作業フローの例である。作業 p1 の成果として a1 を作成し、a1 を input として p2,p3 が並行して行われる。そのとき p2 は a2 という成果物を作り p4 に引き継いで a3 を作成するが、p3 はその作業の中で a3 を作りだす（結合）。a3 を元にして p5 が行われ、最終的には a4 が生成されて作業は終了する。

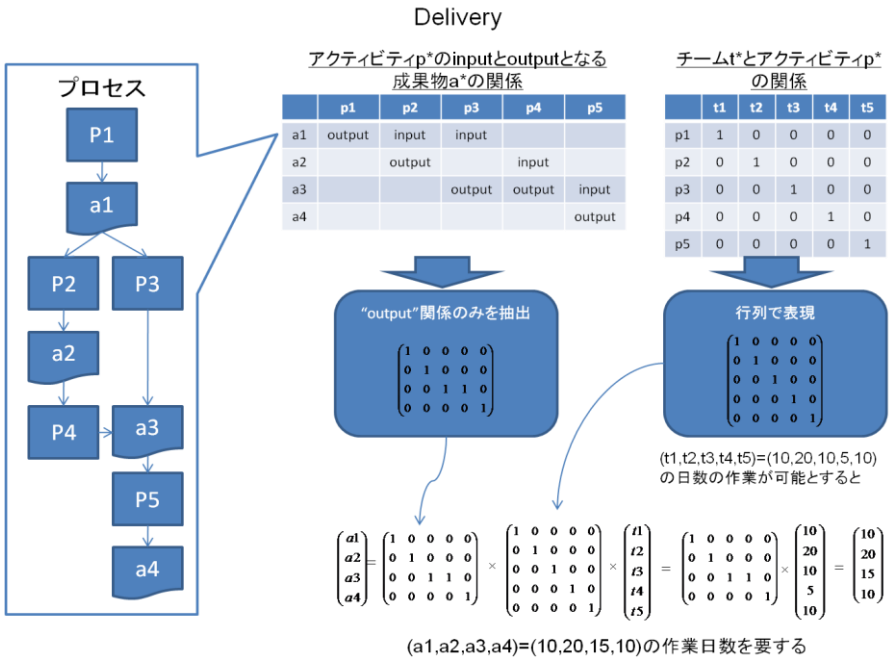


図 43　Delivery 計算のサンプル（2）

この場合も単純化を図るため、チーム t とアクティビティ p の関係は単位行列、つまり互いに独立性を保っているものとした。アクティビティ p とその成果物 a の関係は１：１または２：１の関係性を持っている。結果 team が（t1,t2,t3,t4,,t5）=（10,20,10,5,10）の作業日数があったとすれば（a1,a2,a3,a4）=（10,20,15,10）となり、10+20+15+10=55 日の作業期間を要すると算出される。

基本的な考え方として、サンプル１のアクティビティ p とその成果物 a のマトリックスでみられるように P4/P5 は並行作業が可能であり、その成果物も独立である。このような並行作業を増やすことは作業時間の短縮や、ある一部の変更が他の作業・成果物に対して影響を及ぼすことも少なく、ロバスト性の高い作業の仕組みが確立していると考えることができる。しかし、P2/P3 はフロー図上並行作業として表記されているが、同一成果物を作成するために作業を分担しているに過ぎず、実質的には独立性はなく依存度が高い作業になっている。結果、ロバスト性は低く P2 の作業の影響を P3 は常に意識しておかなければならない。

以上のように、当初述べた「「本来すべき仕事ができていないこと」、「ネットワーク化による複雑化した情報流通」が現在の IT 産業の問題、特に情報システムの高品質化を考える上での課題と考え、その 2 つの課題から見える本質的な問題は「円滑に情報が流通されないこと」である。」ということに関し、システムを構成する要素の関係をネットワークとして捉え、それを行列で表現し分析することの有用性を述べた。それによってコスト、品質、開発期間についても評価することができ、また行列計算のパターン分析で述べたように対角行列が理想形であり、それ以外ならばどの要素関係が全行例で、それをどうコントロールするかがポイントとなることを述べた。そして、どのようなシステムが理想であり、また現在のシステムはどのような問題が潜み、それをどのようにすればよいかをこのマトリックス分析を用いることにより定量的に且つ論理的に考えることが可能になった。

様々な分析を紹介してきたのでここで幾つかの例をまとめ、マトリックス分析を行う上で設定するベクトルの単位は何にすればよいか、それによって行列の成分値が持つ意味も異なる。よって、以下にその代表的なパターンを整理してみた。

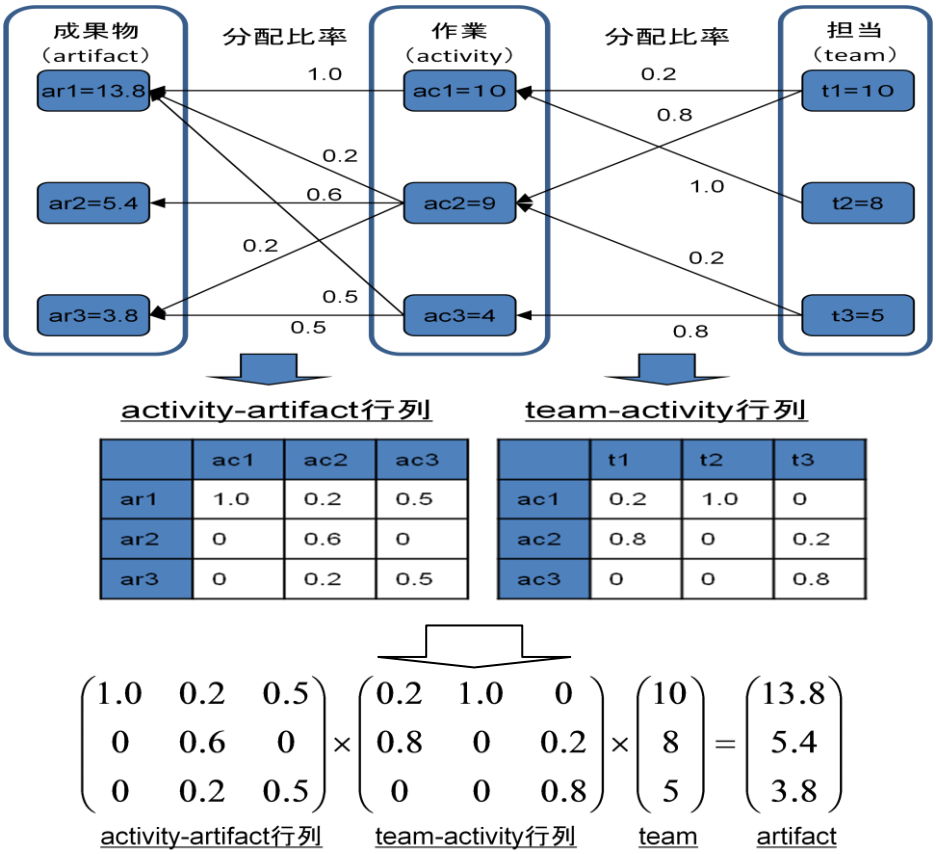
$$\vec{y} = A\vec{x}$$

パターン	Outputとなるベクトル“y“の単位	「行列A」の持つ意味	Inputとなるベクトル“x”の単位
1	作業日数	工数分配率	作業日数
2	原価	原価分配率	原価
3	作業日数	生産性	原価
4	原価	単金	工数
5	売値	単金＋儲け	工数
6	品質	品質	スキル

図 44　マトリックス分析による算出パターン例

このように本研究における行列は複数の意味を持つ。単金を成分として持つ行列の場合もあれば、無単位で工数の分配比率としてしか意味を持っていないケースもある。またデリバリーを考えるときには、日数／人月の単位を持っていることになるであろう。各要素間の関係性は普遍であるが、その行列によって何を求めたいのか、何を分析したいのか、ということによって、その成分値は異なる値を持つことに注意が必要である。

下記の事例では“担当（team）・作業（activity）・成果物（artifact）”の関係を例として説明する。但し、この事例で扱う単位は“人月（工数）”である。



	ac1	ac2	ac3
ar1	1.0	0.2	0.5
ar2	0	0.6	0
ar3	0	0.2	0.5

	t1	t2	t3
ac1	0.2	1.0	0
ac2	0.8	0	0.2
ac3	0	0	0.8

$$\begin{pmatrix} 1.0 & 0.2 & 0.5 \\ 0 & 0.6 & 0 \\ 0 & 0.2 & 0.5 \end{pmatrix} \times \begin{pmatrix} 0.2 & 1.0 & 0 \\ 0.8 & 0 & 0.2 \\ 0 & 0 & 0.8 \end{pmatrix} \times \begin{pmatrix} 10 \\ 8 \\ 5 \end{pmatrix} = \begin{pmatrix} 13.8 \\ 5.4 \\ 3.8 \end{pmatrix}$$

activity-artifact行列　team-activity行列　team　artifact

図 45　行列計算の原理

担当の工数を t1=１０人月、t2=８人月、t3=５人月としたときに、team-activity 行列によって示された工数の分配比率によって各“作業”に割り当てられる工数が決まる。更に各作業（activity）によって作成される成果物（artifact）にかかる工数も activity-artifact 行列によって定義された分配比率によって規定される。このように行列により、元の工数がどのように分配されているか明らかにすることが出来る。よって、各関係を規定する行列をすべて掛け合わせた上で team ベクトルの示す工数を掛ければ各成果物に必要な工数を示す artifact ベクトルを求めることができる。

以下に目的に応じて行列の成分値はどうあるべきか整理した。

- “複雑さ“を求める場合
  - 成分値は“１”とする。
  - 理由として、複雑さは要素の関連性のみを定量化した数値であるから、関連性有り→１、無し→０とすれば十分に目的を満たす。

- “難易度“を求める場合
  - 作業品質を行列成分で示していると考え、基本となる成分値は“１”とする。（作業 a は一生懸命→１、作業 b は手を抜く→０．５などの応用も可能）
- “工数“を求める場合
  - この場合、インプットとなる Team ベクトルの単位も“工数”とすれば、関係性を表す行列成分は工数の配分率を定義したに過ぎない。よって、成分値の列ごとの和は“１”となる。
- “コスト“を求める場合
  - 前提として、requirement ベクトルとしてコストを算出する場合は、現在のソフトウェアの見積においてはコストを算出する定義も定まっていないため、色々なやり方が考えられる。
  - team ベクトルの単位を工数として、行列全体で単金を示すとした場合、個々の行列の持つ意味はさまざまなケースが想定できる。
    - Team-Activity の関係性に単金の意味を持たせる、または Team-Artifact の間（２つの行列を掛け合わせた結果）に単金の意味を持たせるなどのやり方が考えられる。
    - 単金の意味を持たせない行列については、上記の難易度のように作業品質・精度などの無単位の行列を利用する。
    - 最終的は全ての行列を掛け合わせた結果に得られるノルムで単価などを調整することも可能である。
  - team ベクトルの単位に関わらず、requirement ベクトルに関連する要素全てが価格を定義すると考えた場合は、複雑さと同じ行列を用い、全ての行列を掛け合わせた結果に得られる行列のノルムで単価を決定するやり方もある。（本論文ではこの考え方で、計算方法の多くを説明している）

最後に、ＱＣＤの最適解を算出するための一般論を考えてみる。

$\vec{q} = \text{quality} \quad \vec{c} = \text{cost} \quad \vec{d} = \text{delivery}$

$\vec{t}_q = \text{team}\,quality$（組織を構成する各チムの質的要素をベクトル表現したもの）

$\vec{t}_c = \text{teamcost}$（組織を構成する各チムの費用的要素をベクトル表現したもの）

$\vec{t}_d = \text{teamdelivery}$（組織を構成する各チムの活動期間的要素をベクトル表現したもの）

とすると、

$$\vec{q} = A\vec{t}_q$$
$$\vec{c} = B\vec{t}_c$$
$$\vec{d} = C\vec{t}_d$$

表すことができる。これは、ｑ／ｃ／ｄそれぞれの次元がチーム数に依存しており、それぞれの次元が同数を取ることを示している。よって、ｑ／ｃ／ｄの各ベクトルをそれぞれ同様に扱い四則演算することが可能であると言える。

下図のようにｎ次元の空間で各ベクトル（cost, quality, delivery）を軸としたその関係性を考えたときに、そこに出来上がる 4 面体（q-c-d-o）の点 o と平面 q-c-d の距離が最小になっているときが、各値（q、c、d）

の最小値になると考えた。

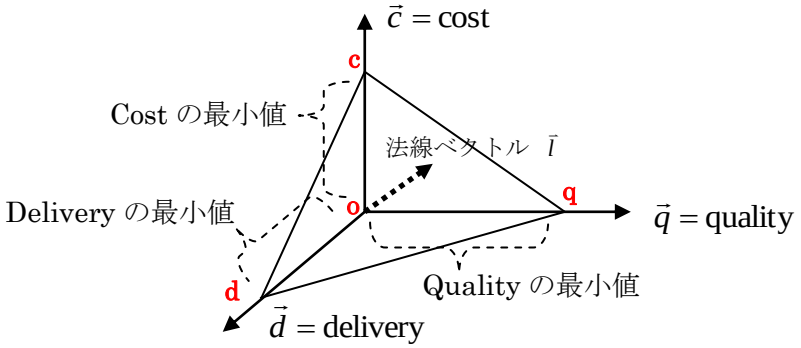


図 46　QCD の最適値評価

よって、まずは平面 q-c-d に対する法線ベクトル（= ）を求め $\vec{l}$、その大きさの最小値を算出することで各項目Ｑ・Ｃ・Ｄの最小値も決定できる。

$$\vec{l} = \vec{q} + \vec{c} + \vec{d} = (a, b, c, d, \cdot\cdot\cdot)$$

とすると、その法線ベクトルの長さ（= l ）は以下のように求めことができ、a,b,c・・の関数となっていることが分かる。

$$l = \left|\vec{q} + \vec{c} + \vec{d}\right| = \sqrt{(a^2 + b^2 + c^2 + d^2 + \cdot\cdot\cdot)}$$

現実的に組織評価で利用する場合は、まず初めにモデルを構成する各ベクトルまたは各行列式で定義されている要素の中で、重要と思われる要素を変数として洗い出すことを行う。そして次に l をその変数（x,y,z・・・）を関数とした方程式として定義し、 l の値が最小になるときの変数を求めればよい。それにより、QCD の最適解を算出することが可能となる。例えば、モンテカルロシミュレーションにより、各変数が取りえる値の範囲で乱数を発生させて l の関数に代入し、その値の変化をみることで l が最小値となる変数を見つけ出すことも可能である。

9.　ケーススタディー

以降では、実際にマトリックス分析を行う場合を想定して、その留意点・分析方法などを解説する。

まず初めに、ネットワーク関連図の作成方法について述べる。

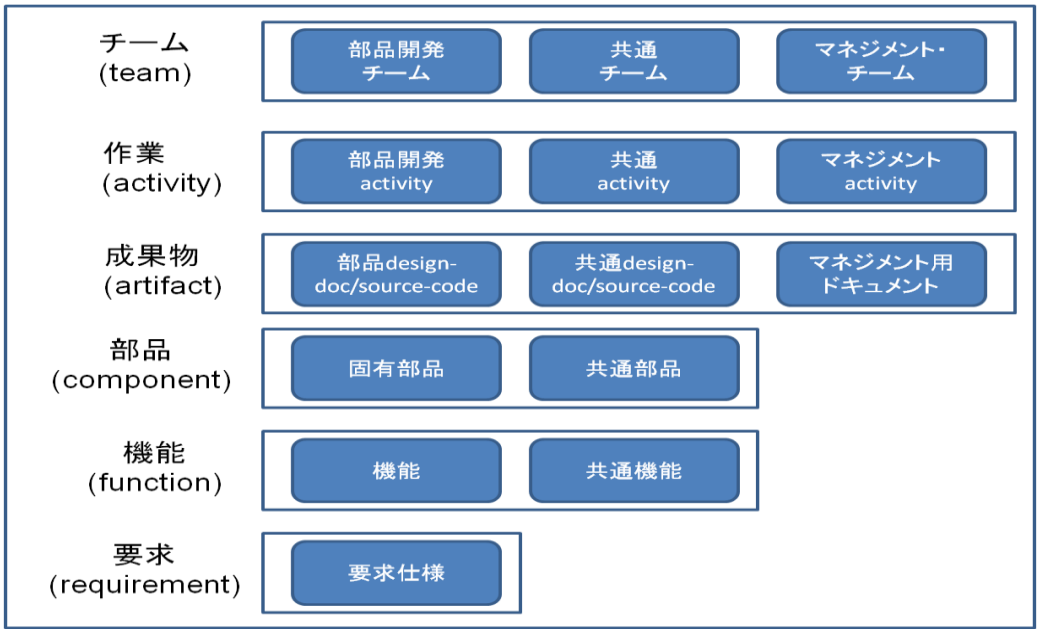


図 47　基本概念の整理

本ネットワーク図を作成するにあたり、基本的な考え方は以下の通りである。

① 作るべき“部品”を中心に考えること

② 作業領域では、時間軸で捉えたプロセスという概念ではなく、部品設計に必要な作業を時間軸に関係なくひとつの塊として扱うこと。

③ 成果物では、設計書や管理ドキュメント、設計規約などは勿論、ソースコードも含む

④ 部品では、モジュール・レベルで考えること

上記基本ルールに対して補足説明を加える。①についてであるが、まず製品やサービスを開発して提供する目標は、設計情報が各作業をとおして設計書に内容をまとめ、そして実際にモジュールとして実装され、要求仕様を満足させるための機能が提供されることである。その全体の仕組みを整理し、その仕組をひとつのシステムとして分析・改善する手法を提案しているのが本研究である。よって、作業プロセスを時系列的に分析するだけで効率化が図れるとは考えていない。その結果が実装される機能にどれだけ満足に実現され、結果として要求仕様をどれだけ満たしているか、ということが重要なのである。その要求仕様を満たすために作る部品をこの仕組の中心として考えて整理することが必要なのである。

先ほども触れたが、そのため作業プロセスを時系列的にリストアップして、作業領域にまとめることは少々ポイントを外していることになる。出来上がる最終結果は同じになるかもしれないが、ある部品を作るために必要な作業は何かという観点で作業を洗い出し、部品との関係性を整理する。そのときに必要な成果物も同時に列挙しておく。はじめに抽出される作業は「〇〇部品開発」というレベルでも構わないかもしれない。最後に、そもそもその中心として考えるべき部品の定義とは何なのか、ある程度その部品が保有する機能として意味を持つレベルの大きさを想定している。例えば、ハードウェアでの話になるがヘアードライヤーならば、そこで使われるネジや熱線などの基本部品を想定していない。それは作業プロセスの中で成果物として出来上が

っているものとする。電機系をひとつのモジュールとして捉えて、電源コードとスイッチをひとつの部品とし、または発熱系の部品をばね、熱伝対、スイッチなどを含めて考えるべきである。自動車では、エンジン、シャーシなどのボディ、その他電機系のエレクトロニクスなどという単位を「部品」と考えるべきである。ソフトウェアで考えてみれば、ひとつの Java クラスを部品と定義するのではなく、複数のクラスをまとめて扱うことが適切であるケースが多いと考えている。パッケージ単位レベルと考えることが適当であろう。

また、上記の表で共通系やマネジメントなどというカテゴリを作ったが、共通系では共通に利用される部品の開発も含んでいるが、直接的には設計として利用はされないが例えば各担当との調整作業や調整用のドキュメント作成なども想定している。つまり、この担当からは必ずしも部品が生成されるとは限らない。また、同様にマネジメントも部品を生成することはない。また、そもそも目指すは単位行列の関係である。独立関係を成り立たせたいので、理想的には共通系などというカテゴリ自体が存在しないことが望ましいとも言える。

次に情報システム開発におけるケーススタディーを行ってみる。

想定する情報システムは下記のユースケースモデルに示すようにログインを可能にし、口座からの引出、預入を可能にする ATM を想定している。

◆　ユースケース

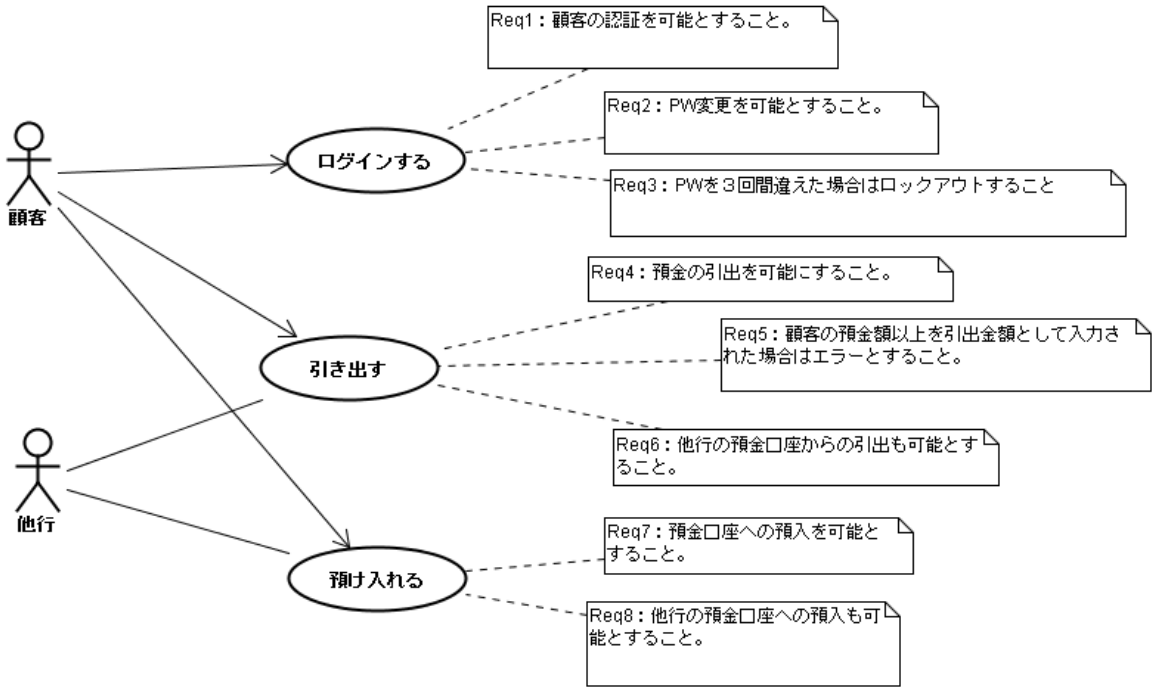


図 48　ケーススタディー　ユースケースモデル

このシステムを利用する立場、つまりシステム外には顧客と他行がある。つまり ATM へアクセスするのは顧客であり、また他行の口座とも連携していることを表している。

まず、ユースケース：ログインする、に対する要求条件は

1. 顧客の認証を可能とすること。
2. パスワード（PW）変更を可能とすること
3. パスワード（PW）を３回間違えた場合はロックアウトすること

ユースケース：引き出す、に対する要求条件は

4. 預金の引出を可能にすること

5. 顧客の預金額以上を引出金額として入力された場合はエラーとすること

6. 他行の預金口座からの引出も可能とすること

ユースケース：預け入れる、に対する要求条件は

7. 預金口座への預入を可能とすること

8. 他行の預金口座への預入も可能とすること

以上のようになっている。

このような要求条件を満たす情報システムを開発するために、開発着手前の企画フェーズではどのようなシステムを作り、それにはどの程度の予算が必要か検討することになる。そこで、この情報システムを開発するプロジェクトをひとつのシステムとして考え、それが構成する要素を洗い出し、その関係性を分析することでマトリックス分析がどのように利用されるか、またその利用結果が正しい解を導いているか検証することとする。企画フェーズの要素間の関連図を示すが、抽出した要素はマトリックスの行列のカテゴリで整理した。整理するに当たっては、UML のクラス図を利用して表記し、そのステレオタイプにマトリックスのカテゴリを示した。

はじめに、team として、プロジェクト管理担当、要求管理担当、アーキテクチャ担当、詳細設計担当、実装担当、テスト担当を挙げた。また activity としては画面設計、業務ロジック設計、DB（データベース）設計を挙げた。そして、artifact としては画面仕様書、画面遷移図、業務フロー図、機能仕様書、他システムインタフェース仕様書、ER 図、ソースコードを挙げている。その結果作られる component としてログイン画面、メニュー画面、引出画面、預入画面、ユーザ管理、引出業務、預入業務、他行連携、ユーザ管理データ、預金管理データという部品を作ることとした。それら部品が保有する機能は、ユーザ ID（UID）入力機能、パスワード（PW）入力機能、メニュー選択機能、引出金額入力機能、預入金額入力機能、UID-PW 管理機能、預金高管理機能、他行連携機能、ユーザ検索機能、引出金額登録機能、預入金額登録機能であり、それらが組み合わさって、要求条件を実現することとする。

各要素を線でつなぐことによって、これらの要素間の関係性を示したのが下図である。

また、その関係性をマトリックス上に●を付けることによって表し、次に●が付いているところを１に、ついいないところ、つまり関係性がないところは０を設定することで行列成分とした行列を作成した。

以降のページでは、これらの関連図、マトリックス、行列を企画フェーズで作成し、そして開発フェーズでは、全ての機能を技術者が開発するのか、または設計書からソースコードを自動生成するようなジェネレータを導入するべきか、パッケージ・ソフト（PKG）を導入するべきかを検討するために、３種類の関係性を図示し評価した。

◆　企画フェーズ（ネットワーク図）

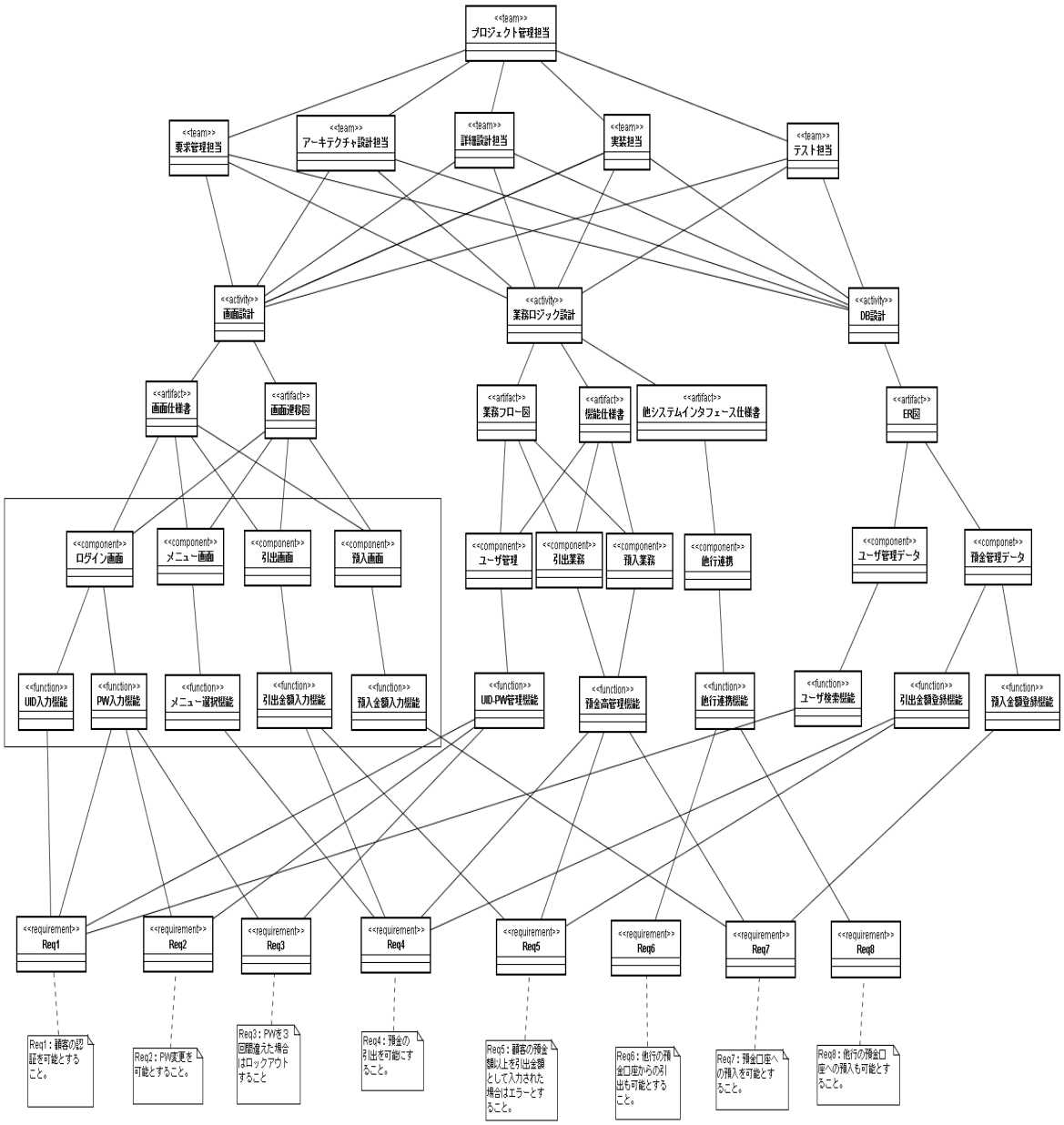


図 49　開発の要素関連図（企画フェーズ）

◆　企画フェーズ（マトリックス）

開発着手前の企画段階では、前述のユースケースモデル程度のもので一度ネットワーク図を作成し、それをベースにマトリックス表を作成してみることをお勧めする。見積根拠や開発標準などを元にすれば作成可能である。また、ネットワーク図はプロジェクトマネジメント的な見地からも開発するシステムの全体像を把握することが可能であり、どの要素に色々なものが絡みネットワーク的なハブになっているか一目瞭然である。
よって、マネジメント上のキーポイントを明確にすることができ、ネットワーク図を用いて関係者とディスカッションし各要素間の関係性を単純化していくことも重要である。
出来上がったネットワーク図をもとに、関係性をマトリックスで表現し直したものを下記に示す。

requirement ＼ function	UID入力機能	PW入力機能	メニュー選択機能	引出金額入力機能	預入金額入力機能	UID-PW管理機能	預金高管理機能	他行連携機能	ユーザ検索機能	引出金額登録機能	預入金額登録機能
顧客の認証を可能とする	●	●				●			●		
PWの変更を可能とすること		●				●					
PWを3回間違えた場合はロックアウトすること		●				●					
預金の引出を可能にすること			●	●			●			●	
顧客の預金額以上を引出金額として入力された場合はエラーとすること				●			●			●	
他行の預金口座からの引出も可能とすること								●			
預金口座への預入を可能とすること					●		●				●
他行の預金口座への預入も可能とすること								●			

function ＼ component	ログイン画面	メニュー画面	引出画面	預入画面	ユーザ管理	引出業務	預入業務	他行連携	ユーザ管理データ	預金管理データ
UID入力機能	●									
PW入力機能	●									
メニュー選択機能		●								
引出金額入力機能			●							
預入金額入力機能				●						
UID-PW管理機能					●					
預金高管理機能						●	●			
他行連携機能								●		
ユーザ検索機能									●	
引出金額登録機能										●
預入金額登録機能										●

component ＼ artifact	画面仕様	画面遷移図	業務フロー図	機能仕様書	他システムインタフェース仕様書	ER図
ログイン画面	●	●				
メニュー画面	●	●				
引出画面	●	●				
預入画面	●	●				
ユーザ管理			●	●		
引出業務			●	●		
預入業務			●	●		
他行連携					●	
ユーザ管理データ						●
預金管理データ						●

artifact ＼ activity	画面設計	業務ロジック設計	DB設計
画面仕様	●		
画面遷移図	●		
業務フロー図		●	
機能仕様書		●	
他システムインタフェース仕様書		●	
ER図			●

activity ＼ 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
画面設計	●	●	●	●	●
業務ロジック設計	●	●	●	●	●
DB設計	●	●	●	●	●

team ＼ team	プロジェクト管理担当
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	●

図 50　マトリックス（企画フェーズ）

要素間の関係性があるところに黒丸●を付けた。これを数学的に処理するために黒丸●を今回は単純に“1”に置き換え、関係性がないところを“0”を設定した行列を下記に作成した。

requirement-function

1	1	0	0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0

function-component

1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1

component-artifact

1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	0	0	1

artifact-activity

1	0	0
1	0	0
0	1	0
0	1	0
0	1	0
0	0	1

activity-team

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

team-team

1
1
1
1
1

図 51　行列（企画フェーズ）

全行列を掛け算し、この組織体の性質を示す行列とする。その結果が下記であり、

7	7	7	7	7
4	4	4	4	4
4	4	4	4	4
9	9	9	9	9
7	7	7	7	7
1	1	1	1	1
7	7	7	7	7
1	1	1	1	1

⇒

7	7	7	7	7	0	0	0
4	4	4	4	4	0	0	0
4	4	4	4	4	0	0	0
9	9	9	9	9	0	0	0
7	7	7	7	7	0	0	0
1	1	1	1	1	0	0	0
7	7	7	7	7	0	0	0
1	1	1	1	1	0	0	0

図 52　行列計算結果と正方行例化（企画フェーズ）

ノルム＝36.193922

と算出できる。内積を算出する場合は、対角行列のロバスト性が一番高いという公理から、それを基準に考えて、成分値＝０として正方行列化して計算する（成分値＝０ならばノルムには影響がないため、ノルム値はそのまま計算して問題ない）。　以降の計算には前述の Scilab を利用した。

固有ベクトルの算出結果は

a　=(0. ,0. ,0. ,0. ,0. ,1. ,0. ,0.)

b　=(0. ,0. ,0. ,0. ,0. ,0. ,1. ,0.)

c　=(0. ,0. ,0. ,0. ,0. ,0. ,0. ,1.)

d　=(0.2824970 ,0.1614269 ,0.1614269 , 0.8878477 ,0.2824970 , 0.0125 ,0. ,0.0125)

e　=(0.4907371 ,0.2804212, 0.2804212, 0.6309476, 0.4907371 ,0.0701053 ,0.4907371, 0.0701053)

f　=(0.8011155 ,0.1136483 ,0.1136483 ,0.0054642 ,0.5792830 ,0.0095493 ,0.1883295 ,0.0095493)

g　=(0.1836862 ,0.8950365 ,0.1049635 ,0.2053300 ,0.4010568 ,0.0068726 ,0.0179855 ,0.0068726)

h　=(0.1836862 ,0.1049635 ,0.8950365 ,0.2053300 ,0.4010568 ,0.0068726 ,0.0179855 ,0.0068726)

となる。

この固有ベクトルに対応する固有値は

‘a⇒0

‘b⇒0

‘c⇒0

‘d⇒4.441D-16

‘e⇒31

‘f⇒0

‘g⇒0

‘h⇒0

であるから、固有ベクトルの和は

i=a*0+b*0+c*0+d*4.441D-16+e*31+f*0+g*0+h*0

i　=[15.212849 , 8.6930565, 8.6930565, 19.559377, 15.212849, 2.1732641, 15.212849, 2.1732641]

となる。

また、単位行列の固有値・固有ベクトルから求められるベクトル和は

j=[1,1,1,1,1,1,1,1]

であるから、その大きさは

sqrt(j*j') =　2.8284271

となる。

よって、このベクトルと単位行列から導かれる固有値・固有ベクトルの和の間で作られる空間における内積は

cosθ＝86.930565/(35.177353*2.8284271) = 0.8737041

となり、よって、

品質＝1/norm×cosθ＝0.8737041/36.193922 = <u>0.0241395</u>

と計算することができる。

◆　開発フェーズースクラッチ開発（ネットワーク図）

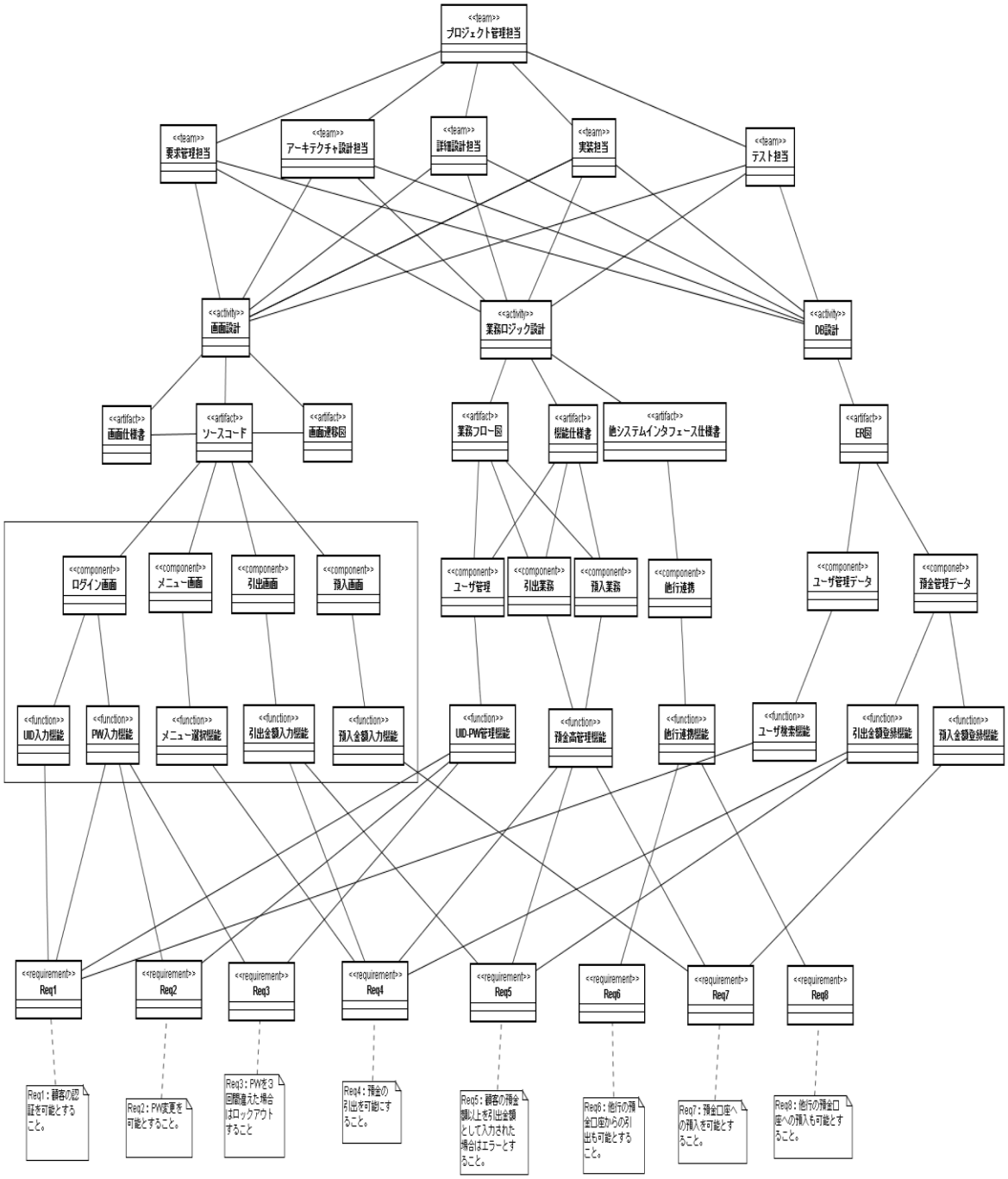


図 53　開発の要素関連図（スクラッチ開発）

◆　開発フェーズースクラッチ開発（マトリックス）

これも先ほどと同様にネットワーク図をマトリックスで置き換えてみる。特に開発に着手する前後ではより精緻な検討が必要である。よって、以降ではパッケージソフトを導入する場合や自動でソースコードを生成するジェネレータを利用する場合、そして、各技術者がすべて自前でソースコードを作成する場合の３パターンを考えた。

まずここでは全てのソースコードを技術者が手書きするケースを想定した。前述のネットワーク図を下記のようにマトリックスで表現し直し、そして更に０，１で数値化し行列を作成した。

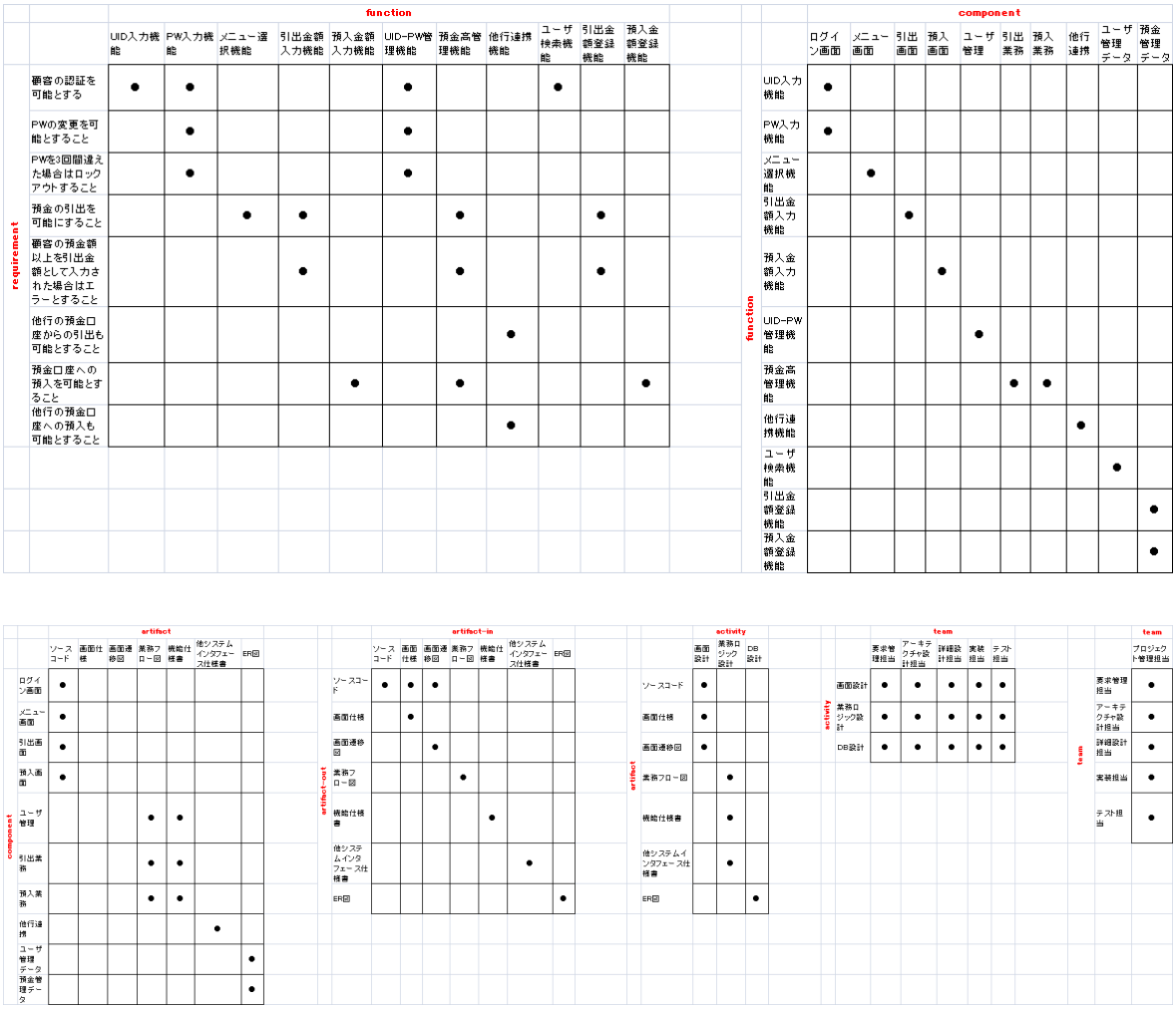


図 54　マトリックス（スクラッチ開発）

行列を作成する際には、各成分値を何にすべきか検討すべきである。例えば利用する技術の難易度を加味する必要もあるまたは技術者のスキルも加味する必要もある。そのような個々の要素の難易度を数値表現することで、より現実を正確に描写した行列を作成することができる。具体的に設定する値については、個々に基準を定義しルール付けする中で設定すれば良いが、概算レベルで検討する場合にはそこまで厳密にしなくてもマネジメントチームにより決定した値を設定するだけで十分と考える。

なお、設定する値は、品質を議論するためなのか生産性を議論するためなのかなど、その目的に応じて異なるが、基準となる値は単位行列の成分である“１”が基準である。

今回は説明をシンプル化するために全ての値を“１”と設定している。

requirement-function

1	1	0	0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0

function-component

1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1

component-artifact

1	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	0	1	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	0	1

artifact-artifact

1	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1

artifact-activity

1	0	0
1	0	0
1	0	0
0	1	0
0	1	0
0	1	0
0	0	1

activity-team

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

team-team

1
1
1
1
1

図 55　行列（スクラッチ開発）

この行列を全部掛け合わせた結果が下記になり、それを正方行列化した結果を示す。

9	9	9	9	9
5	5	5	5	5
5	5	5	5	5
11	11	11	11	11
8	8	8	8	8
1	1	1	1	1
8	8	8	8	8
1	1	1	1	1

⇒

9	9	9	9	9	0	0	0
5	5	5	5	5	0	0	0
5	5	5	5	5	0	0	0
11	11	11	11	11	0	0	0
8	8	8	8	8	0	0	0
1	1	1	1	1	0	0	0
8	8	8	8	8	0	0	0
1	1	1	1	1	0	0	0

図 56　行列計算結果と正方行例化（スクラッチ開発）

この正方行列化した結果、先ほどの事例と同様に固有値・固有ベクトルを計算し、その cos θ やノルムを求めて品質を算出する。

結果は

ノルム（norm）=43.703547

であり、

cosθ＝　104.48644/(42.54518*2.8284271)= 0.8682896

よって、

品質＝1/norm×cosθ＝0.8682896/43.703547= 0.0198677

◆　開発フェーズ－ＰＫＧ利用

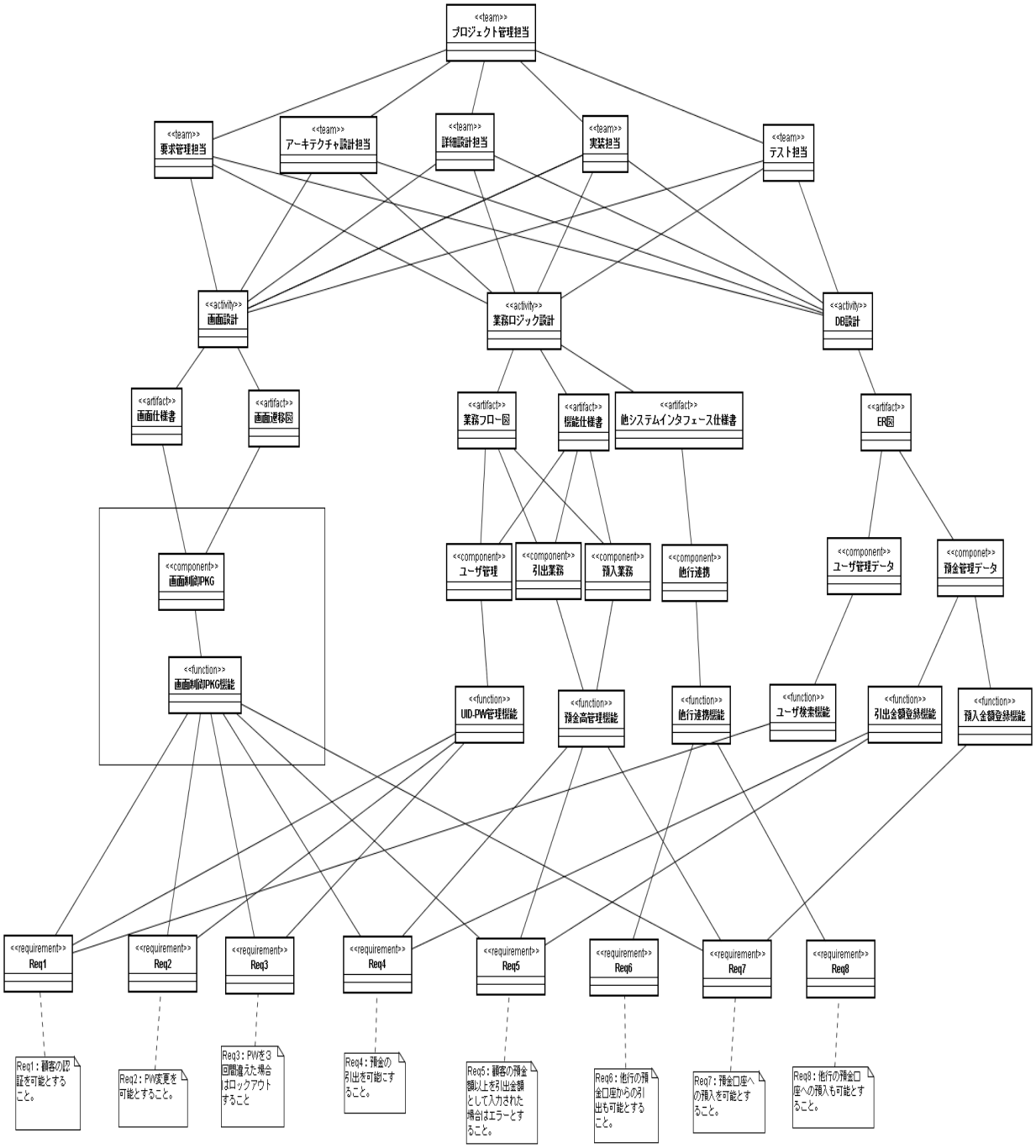


図 57　開発の要素関連図（ＰＫＧ利用）

◆　開発フェーズーＰＫＧ利用（マトリックス）

今までの事例と同様にネットワーク図をマトリックス化した。

requirement \ function	画面生業PKG機能	UID-PW管理機能	預金高管理機能	他行連携機能	ユーザ検索機能	引出金額登録機能	預入金額登録機能
顧客の認証を可能とする	●	●			●		
PWの変更を可能とすること	●	●					
PWを3回間違えた場合はロックアウトすること	●	●					
預金の引出を可能にすること	●		●			●	
顧客の預金額以上を引出金額として入力された場合はエラーとすること	●		●			●	
他行の預金口座からの引出も可能とすること				●			
預金口座への預入を可能とすること	●		●				●
他行の預金口座への預入も可能とすること				●			

function \ component	画面制御PKG	ユーザ管理	引出業務	預入業務	他行連携	ユーザ管理データ	預金管理データ
画面制御PKG機能	●						
UID-PW管理機能		●					
預金高管理機能			●	●			
他行連携機能					●		
ユーザ検索機能						●	
引出金額登録機能							●
預入金額登録機能							●

component \ artifact	画面仕様	画面遷移図	業務フロー図	機能仕様書	他システムインタフェース仕様書	ER図
画面制御PKG	●	●				
ユーザ管理			●	●		
引出業務			●	●		
預入業務			●	●		
他行連携					●	
ユーザ管理データ						●
預金管理データ						●

artifact \ activity	画面設計	業務ロジック設計	DB設計
画面仕様	●		
画面遷移図	●		
業務フロー図		●	
機能仕様書		●	
他システムインタフェース仕様書		●	
ER図			●

activity \ 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
画面設計	●	●	●	●	●
業務ロジック設計	●	●	●	●	●
DB設計	●	●	●	●	●

team \ team	プロジェクト管理担当
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	●

図 58　マトリックス（PKG 利用）

マトリックスで整理した関係性を行列で表すと下記のようになる。

requirement-function

1	1	0	0	1	0	0
1	1	0	0	0	0	0
1	1	0	0	0	0	0
1	0	1	0	0	1	0
1	0	1	0	0	1	0
0	0	0	1	0	0	0
1	0	1	0	0	0	1
0	0	0	1	0	0	0

function-component

1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	1	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	0	1

component-artifact

1	1	0	0	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	0	0	1

artifact-activity

1	0	0
1	0	0
0	1	0
0	1	0
0	1	0
0	0	1

activity-team

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

team-team

1
1
1
1
1

図 59　行列（PKG 利用）

この行列を全て掛け合わせ、正方行列化すると

5	5	5	5	5
4	4	4	4	4
4	4	4	4	4
7	7	7	7	7
7	7	7	7	7
1	1	1	1	1
7	7	7	7	7
1	1	1	1	1

5	5	5	5	5	0	0	0
4	4	4	4	4	0	0	0
4	4	4	4	4	0	0	0
7	7	7	7	7	0	0	0
7	7	7	7	7	0	0	0
1	1	1	1	1	0	0	0
7	7	7	7	7	0	0	0
1	1	1	1	1	0	0	0

図 60　行列計算結果と正方行例化（スクラッチ開発）

となる。

よって、また同様にノルムや cos θ を算出すると以下のようになる。

ノルム（norm）=32.093613

cosθ＝78.654728/(31.358548*2.8284271)=　0.8867964

結果、品質値は

品質＝1/norm×cosθ

＝0.8867964/32.093613

= <u>0.0276316</u>

である。

◆　開発フェーズージェネレータ（ネットワーク図）

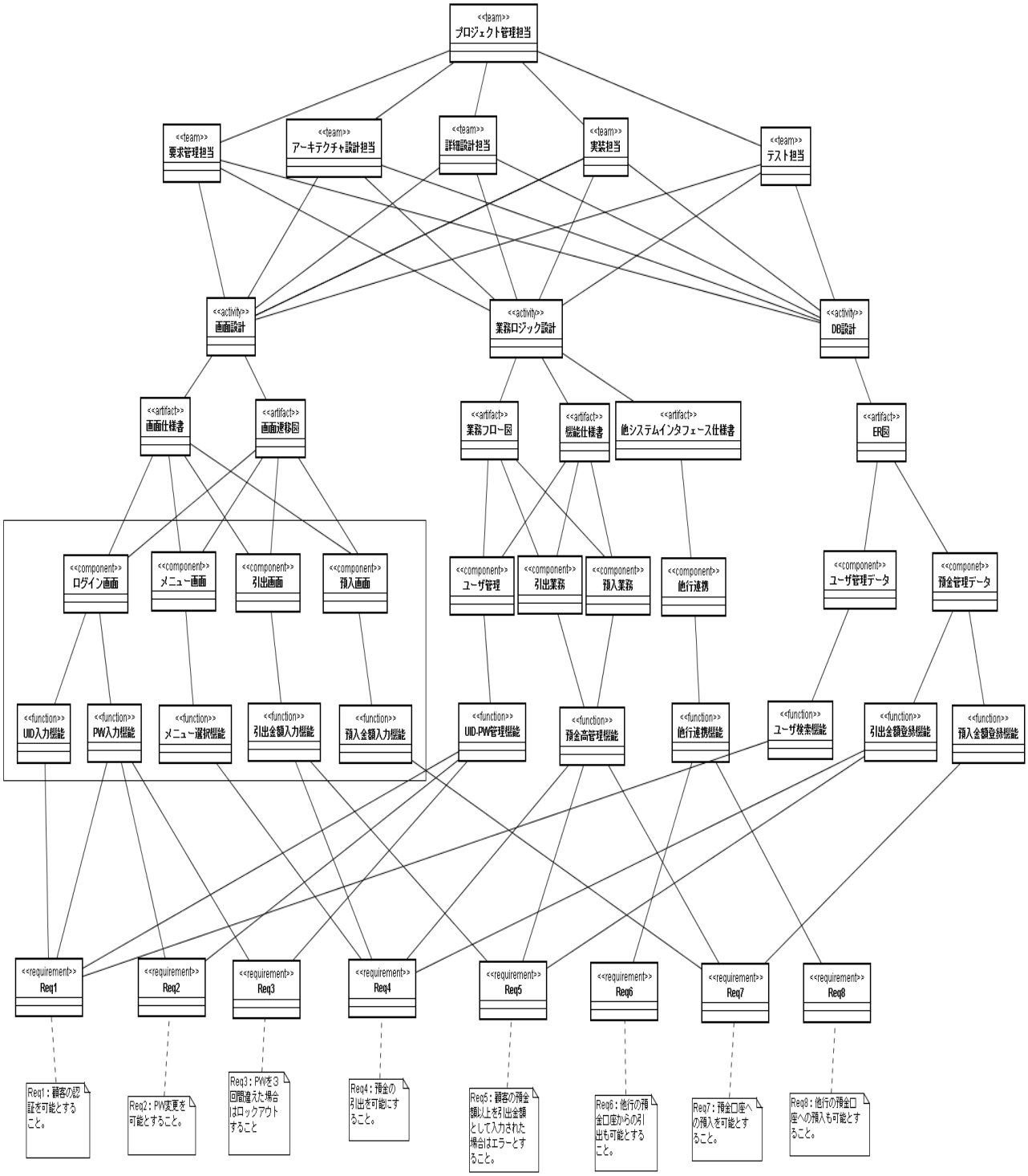


図 61　開発の要素関連図（ジェネレータ利用）

◆　開発フェーズージェネレータ（マトリックス）

最後の事例である。これも同様にマトリックス化し 0,1 の数値化した行列を示す。

requirement \ function	UID入力機能	PW入力機能	メニュー選択機能	引出金額入力機能	預入金額入力機能	UID-PW管理機能	預金高管理機能	他行連携機能	ユーザ検索機能	引出金額登録機能	預入金額登録機能
顧客の認証を可能とする	●	●				●			●		
PWの変更を可能とすること		●				●					
PWを3回間違えた場合はロックアウトすること		●				●					
預金の引出を可能にすること			●	●			●			●	
顧客の預金額以上を引出金額として入力された場合はエラーとすること				●			●			●	
他行の預金口座からの引出も可能とすること								●			
預金口座への預入を可能とすること					●		●				●
他行の預金口座への預入も可能とすること								●			

function \ component	ログイン画面	メニュー画面	引出画面	預入画面	ユーザ管理	引出業務	預入業務	他行連携	ユーザ管理データ	預金管理データ
UID入力機能	●									
PW入力機能	●									
メニュー選択機能		●								
引出金額入力機能			●							
預入金額入力機能				●						
UID-PW管理機能					●					
預金高管理機能						●	●			
他行連携機能								●		
ユーザ検索機能									●	
引出金額登録機能										●
預入金額登録機能										●

component \ artifact	画面仕様	画面遷移図	業務フロー図	機能仕様書	他システムインタフェース仕様書	ER図
ログイン画面	●	●				
メニュー画面	●	●				
引出画面	●	●				
預入画面	●	●				
ユーザ管理			●	●		
引出業務			●	●		
預入業務			●	●		
他行連携					●	
ユーザ管理データ						●
預金管理データ						●

artifact \ activity	画面設計	業務ロジック設計	DB設計
画面仕様	●		
画面遷移図	●		
業務フロー図		●	
機能仕様書		●	
他システムインタフェース仕様書		●	
ER図			●

activity \ 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
画面設計	●	●	●	●	●
業務ロジック設計	●	●	●	●	●
DB設計	●	●	●	●	●

team \ team	プロジェクト管理担当
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	●

図 62　マトリックス（ジェネレータ利用）

requirement-function

1	1	0	0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0

function-component

1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1

component-artifact

1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	0	0	1

artifact-activity

1	0	0
1	0	0
0	1	0
0	1	0
0	1	0
0	0	1

activity-team

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

team-team

1
1
1
1
1

図 63　行列（ジェネレータ利用）

全行列を掛け合わせた結果と正方行列化したものは以下の通りである。

7	7	7	7	7
4	4	4	4	4
4	4	4	4	4
9	9	9	9	9
7	7	7	7	7
1	1	1	1	1
7	7	7	7	7
1	1	1	1	1

7	7	7	7	7	0	0	0
4	4	4	4	4	0	0	0
4	4	4	4	4	0	0	0
9	9	9	9	9	0	0	0
7	7	7	7	7	0	0	0
1	1	1	1	1	0	0	0
7	7	7	7	7	0	0	0
1	1	1	1	1	0	0	0

図 64　行列計算結果と正方行例化（ジェネレータ利用）

これまでと同様にノルム、cos θ を算出すると

ノルム（norm）=36.193922

cosθ＝86.930565/(35.177353*2.8284271)

= 0.8737041

よって、

品質＝1/norm×cosθ

＝0.8737041/36.193922

= 0.0241395

となる。

◆　比較分析（コスト、生産性）

前頁までのケースについて、コスト・生産性について分析を行う。チームベクトルの単位を工数として各チームが 1 人月の工数を要すると仮定した。そのときマトリックスが生産性を表すと考え、算出される requirement ベクトルは各要求の価格と考える。

	requirement								
	顧客の認証を可能とする	PWの変更を可能とすること	PWを3回間違えた場合はロックアウトすること	預金の引出を可能にすること	顧客の預金額以上を引出金額として入力された場合はエラーとすること	他行の預金口座からの引出も可能とすること	預金口座への預入を可能とすること	他行の預金口座への預入も可能とすること	合計
ジェネレーター	35	20	20	45	35	5	35	5	200
PKG	25	20	20	35	35	5	35	5	180
スクラッチ	45	25	25	55	40	5	40	5	240

図 65　コスト比較

①ジェネレータの場合　：　5 人月の工数で、200 万円の仕事

②PKG 利用の場合　　：　5 人月の工数で、180 万円の仕事

③スクラッチ開発の場合：　5 人月の工数で、240 万円の仕事

となり、生産性では 36 万／人月（PKG）＜48 万／人月（スクラッチ）という関係式が成り立ち、PKG が一番良いことが分かる。このように同じ工数をかけてスクラッチ開発の生産性が一番悪いということが分かる。PKG 利用時はスクラッチ開発にくらべて 25%ほど低減され、生産性が高い。但し、この試算には PKG のライセンス価格や技術者のラーニングカーブ（習熟コスト）、またハードウェア価格も含まれていないことに注意が必要である。

また今回は各要素のマッピングを行列で表現するときに成分をすべて“1”として、そこから算出された値を単純に価格として解釈してしまった。本来ならば、行列の各成分値が実際に工数として、または金額として幾ら程度に該当するのか、指標値（単価）を決めて議論することが必要である。

◆　比較分析（品質）

次に、品質について計算した結果を述べる。下表のように複雑性を内積から求められる cos θ の値を用いて品質を算出すると、対角行列である単位行列が基準値であるので、スクラッチ開発が一番低い値を示している。

	複雑性＝cos θ	難易度＝1/ノルム	品質＝複雑性x難易度
ジェネレーター	0.873704	1/36.193922	0.024140
PKG	0.886796	1/32.093613	0.027632
スクラッチ	0.868290	1/43.703547	0.019868

図 66　品質の比較

複雑性、難易度を個別に見てみると、やはり関係図上でも一番複雑な関係図となっているスクラッチが値として低く、次がジェネレータとなっている。また特に明らかな差が付いているのが難易度であり、スクラッチ開発が明らかに高い数値を示し開発の難しさを示している。よって、複雑性、難易度共に低いポイントを取っているスクラッチ開発が品質として一番低く、PKG 開発は逆に複雑性、難易度ともに高いポイントを取っていることによって、品質も一番高いものとなっている。

また、今回品質を計算する場合には、前述のとおりに 3 列追加して成分値＝0 として正方行列化して計算したことに注意が必要である。このように成分値＝0 を追加した非正則行列では、対角行列を基準とした考え方（ロバスト性が一番高いという公理）から、このように 0 の成分を追加することによって正方行列化することが妥当と考えた。

7	7	7	7	7	0	0	0
4	4	4	4	4	0	0	0
4	4	4	4	4	0	0	0
9	9	9	9	9	0	0	0
7	7	7	7	7	0	0	0
1	1	1	1	1	0	0	0
7	7	7	7	7	0	0	0
1	1	1	1	1	0	0	0

図 67　正方行列化したマトリックス

参考までに、上記のように成分＝0 の行・列を増やすことで複雑性・難易度そして品質に対してどのような影響を及ぼしているのか、シミュレーションによって調査を行った。

以下にそのシミュレーション状況を示す。ノルムは変化しないため難易度に変化はないが、行・列を追加したために冗長性が増し複雑性が増していることが分かる。結果として品質にもその影響があらわれている。

但し、複雑性つまり、内積から導き出した cos θ の値は揺らぎながら減少している。この cos θ の意味するところは正方行列の固有値・固有ベクトルで作られるベクトル空間と単位行列の固有値・固有ベクトルで作られるベクトル空間との歪みを示したものに等しい。

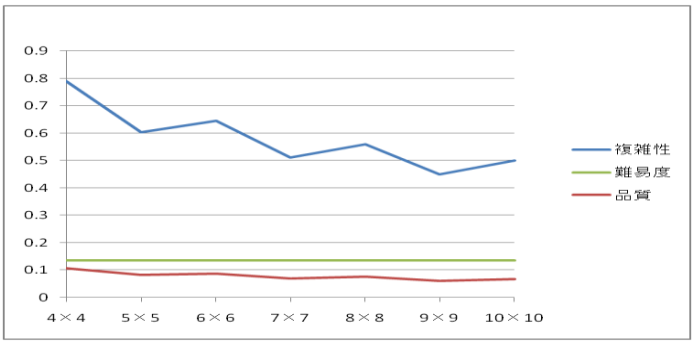
行列の次数が増加すれば、当然その分干渉し合う要素が増えることになるので、その複雑性が増すことになり、結果として、複雑性を示すグラフの傾向が１より減少していくことは理にかなっていると考えている。しかし、このように複雑性を示す値が揺らぎながら減少していく理由は明確ではない。以降では、その原因についての考察を述べる。

0	0	0	0	0	0	0	0	0	0	10×10
0	0	0	0	0	0	0	0	0	0	9×9
0	0	0	0	0	0	0	0	0	0	8×8
0	0	0	0	0	0	0	0	0	0	7×7
0	0	0	0	0	0	0	0	0	0	6×6
0	0	0	0	0	0	0	0	0	0	5×5
0	0	0	0	0	0	0	0	0	0	4×4
0	0	0	0	0	0	2	1	0	0	
0	0	0	0	0	0	3	2	1	1	
0	0	0	0	0	0	0	0	4	6	

元々の行列に対して、行・列共に成分値“０”を追加して正方行列化を行った。

図 68　シミュレーション用の行列

以下がシミュレーションの結果をグラフ化した結果である。



マトリックスの次数変化と難易度・複雑性（=品質）の関係

	4×4	5×5	6×6	7×7	8×8	9×9	10×10
複雑性	0.7903308	0.6039664	0.645302	0.510445	0.558848	0.45017	0.499849
難易度	0.13532105	0.13532105	0.135321	0.135321	0.135321	0.135321	0.135321
品質	0.1069484	0.081729	0.087323	0.069074	0.075624	0.060918	0.06764

図 69　正方行列化とその影響のシミュレーション

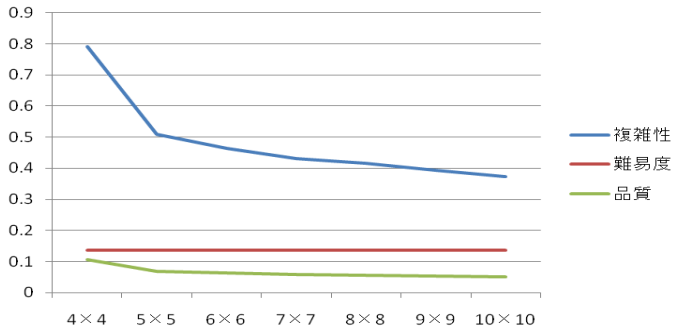
このように複雑性についてはその値が上下しながら全体的には減少する傾向にある。奇数次の行列では減少し、偶数次の行列では増加するという具合である。

そこで、“摂動”という概念を導入する。具体的に説明すると、計算の対象としている行列は各要素の関係性を示している。そのように考えると、次数が増加したことは行列の対象としているシステムを構成する要素が増加したことを意味する。現実社会では構成要素が増加したにもかかわらず、その関係性が全くないこと、つまり成分値が“０”になることは考えにくい。よって、微少な数値を“０”の代わりに設定してみることを試みた。

今回は“0”の代わりに“0.000000001“を代入することにより、上記と同様に難易度や複雑性を算出した。結果は以下のとおり、複雑性は揺らぐこともなく減少傾向を示している。また微少値を設定しただけであるためその難易度については変化が見られない。

現実的なモデルとしては今回示したように、複雑性については減少傾向を示すべきであり、揺らぎ傾向を示

すことは現実的ではない。よって、意図的に正方行列化して、次数変化を行うような場合は、“０”を成分として設定するよりも摂動として非常に微少な値を成分として設定するほうがよいことが分かった。



摂動理論反映ーマトリックスの次数変化と難易度・複雑性（=品質）の関係

	4×4	5×5	6×6	7×7	8×8	9×9	10×10
複雑性	0.790331	0.509336	0.464543	0.430095	0.416226	0.392421	0.372284
難易度	0.135321	0.135321	0.135321	0.135321	0.135321	0.135321	0.135321
品質	0.106948	0.068924	0.062862	0.058201	0.056324	0.053103	0.050378

図 70　摂動を取り入れた場合の複雑性と難易度、品質

ケーススタディーでは、全て 8×8 の行列であり、その行列で比較分析を行った。このような場合は成分値=０としても、互いの行列の条件が同じであるために分析には影響ないと考える。しかし、8×8 の行列で表現されるシステムの品質と 5×5 の行列で表現されるシステムの品質を比較する場合などは、摂動を考慮して複雑性を算出しないと、前述のグラフで示したとおりに複雑性はその値に揺らぎがみられるために正しい評価を行うことが難しい。

行列の次数が異なったもの同士を比較分析したり、行列の次数を変化させて互いを比較評価行ったりする場合などには、以上のような観点を考慮し必要に応じて摂動の考え方を用い、微少値を設定して計算を行う必要があると言える。

◆　比較分析（開発期間）

まずは、ジェネレータ利用の時と PKG 利用時の artifact－activity の関係を整理する。

	artifact-activity		
	画面設計	業務ロジック設計	DB設計
画面仕様	out	0	0
画面遷移図	out	0	0
業務フロー図	0	out	0
機能仕様書	0	out	0
他システムインタフェース仕様書	0	out	0
ER図	0	0	out

図 71　ジェネレータ利用時のマップ

	artifact-activity		
	画面設計	業務ロジック設計	DB設計
画面仕様	out	0	0
画面遷移図	out	0	0
業務フロー図	0	out	0
機能仕様書	0	out	0
他システムインタフェース仕様書	0	out	0
ER図	0	0	out

図 72　PKG 利用時のマップ

このように横軸の activity で作成される artifact に“out”と記載し、その関連付けをマップ化している。このマップを数値 0/1 を用いて行列化する。Out=1 として、それ以外は 0 の行列成分とする。

1	0	0
1	0	0
0	1	0
0	1	0
0	1	0
0	0	1

1	0	0
1	0	0
0	1	0
0	1	0
0	1	0
0	0	1

図 73　ジェネレータ利用時の行列（左）と PKG 利用時の行列（右）

この artifact－activity の関係行列を team ベクトルと activity－team 行列の掛け算を行うことで各生産物を作成するのに掛かる作業日数を算出する。つまり team ベクトルが各メンバの工数とすれば、作業日数を算出する行列（artifact－activity×activity－team）は生産性を表す行列ということになる。

ジェネレータ利用時の計算はこのようになり、artifact のベクトルとして（5,5,5,5,5,5）という値が得られ、各 artifact の作成にかかる日数がそれぞれ 5 日ということとなる。

artifact－activity　　activity－team　　team

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \end{pmatrix}$$

図 74　ジェネレータ利用時の行列計算（成果物作成にかかる作業日数の算出）

また、同様に PKG 利用時の場合は以下のようになる。

$$\overset{\text{artifact－activity}}{\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}} \times \overset{\text{activity－team}}{\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}} \times \overset{\text{team}}{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} = \begin{pmatrix} 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \end{pmatrix}$$

図 75　PKG 利用時の行列計算（成果物作成にかかる作業日数の算出）

また、デリバリーの最短値を算出するには、上記で算出した作業日数の和を取るだけでは求めることができず、作業プロセスのフローを考える必要がある。作業フローを検討して、並列作業するところもあれば、シーケンシャルにしか作業ができないケースもある。

そのように最短の作業日数を求める場合には作業フローを別途考えることが必要であり、その場合はたとえば DSM などを用いて作業プロセスの分析などを明らかにすればよい。

次にスクラッチで開発する場合について検証を行う。このパターンの場合は生産物間に依存関係があるために生産物（artifact）間の関係性も考慮しなければならない。

成果物間の関係性を示したマトリックスでは画面仕様書と画面遷移図からソースコードが生成されているという依存関係が表現されている点が特徴である。マトリックスの記述ルールとして、上三角形の領域には成果物の関係上、input となる情報に対して印を付け、下三角形の領域には output となる情報に対して印を付けることとした。

		input						
		ソースコード	画面仕様	画面遷移図	業務フロー図	機能仕様書	他システムインタフェース仕様書	ER図
output	ソースコード	out	in	in	0	0	0	0
	画面仕様	0	out	0	0	0	0	0
	画面遷移図	0	0	out	0	0	0	0
	業務フロー図	0	0	0	out	0	0	0
	機能仕様書	0	0	0	0	out	0	0
	他システムインタフェース仕様書	0	0	0	0	0	out	0
	ER図	0	0	0	0	0	0	out

図 76　artifact－artifact 間の関係性

本例では上記のとおり画面仕様と画面遷移図が input 情報として、ソースコードと関係付けているために“in“という印を上三角形に記入している。

各成果物の作成にかかる作業日数の算出にあたっては、今までと同様に行列の掛け算を行うが、このような場合は artifact－artifact と関係を示す行列も含めて掛け算を行う必要がある。計算結果は以下の通りである。

$$\overset{\text{artifact－artifact}}{\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}} \times \overset{\text{artifact－activity}}{\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}} \times \overset{\text{activity－team}}{\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}} \times \overset{\text{team}}{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} = \begin{pmatrix} 15 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \end{pmatrix}$$

図 77　スクラッチ開発時の行列計算（成果物作成にかかる作業日数の算出）

このように各成果物にかかわる作業日数を算出することができるが、前述のとおり、作業プロセス全体の最短日数を知りたい場合は他のアプローチ（DSM）などを用いた分析する必要がある。

◆　ケーススタディーまとめ（情報伝達の円滑度）

以上のように、感覚的には常識の範疇であったことが本モデルによる理論計算でも示すことができた。パッケージ（PKG）が一番効率的であること、品質も高いこと、など常識的なことではあるが、それを定量的に説明する理論は今までなかったが、この手法とモデルを用いることで定量的な議論が可能になった。

例えば本アプローチにより、PKG を利用するとスクラッチ開発で開発するよりも大凡どの作業が軽減され、結果的に何パーセントの生産性向上効果が見込まれるのか、理論計算で示すことが可能である。今回の結果から、スクラッチでの開発よりも自動でソースコードを生成するジェネレータでの開発は 17%程度の生産性向上効果があるとの算出結果が得られる。筆者は過去に開発作業として定義されている WBS ごとに積み上げることによって、同様なジェネレータを利用した場合の生産性向上効果を算出したことがある（榮谷昭宏（2005））。その結果は約 18%であったことを考えると、たとえ要求条件が異なるにしても各構成要素を繋いだネットワーク図は類似しており、その結果となる算出値が非常に近い値を示したことは、この算出方法が全く当てにならないものではないことは証明されたと考える。

このような生産性向上効果の見積もりは旧来のソフトウェア工学上でも中々難しい分野であり、一般的に利用されるまでに至った手法はないように考える。今回の手法により、その課題を解決する突破口になれば幸いである。

また、品質値については、ここで言う“品質”とは机上レビューや結合試験などの結果で示される品質とは異なるが、イメージは掴んで頂けたであろうか。出来上がったソフトウェアのみの品質ではなく、それを作る組織力全体を含んだ品質なのである。よって、行列の次数が増せば品質値も低下傾向を示すし、また対角行列ではなく三角行列や全行列に近い行列関係の方が同様に低い品質を示す。公理的設計で述べられている独立公理と情報公理の２つを満たしたものが一番高い品質値を示すことになる。各要素の関係性が対角化することこそが設計の目指すべきところであり、それが高品質を生み出す仕組みのプリンシプル（原則）であると言える。ルール以前にこのプリンシプルに近づける事こそが重要なのである。

本来ならば開発組織全体の設計作業の質を向上することこそが、出来上がる情報システムの品質向上に繋がる。しかし現在多くのシステム開発では、“成果物”のみに品質評価の重点を置いている。そのため、品質評価観点から漏れたチェックは行われず、本質的な問題が残存したまま運用する情報システムも多い。無限の時間とコストを費やして全網羅的なテストを設計書や最終成果物に対して実施することも、手段としてはあるがその実効性は非常に低く、非効率である。

組織活動そのものを評価することも CMMI(capability maturity model integration)により実施されるようにはなってきたが、CMMI では作業が定型化し、その習熟度が上がれば高い評価が得られてしまい、本来その定型化した作業のやり方が最善なのかどうかということまで踏み込んだ評価指標を持っている訳ではない。

しかし本研究で述べた方法論では、組織体の品質を評価し如何に対角化（To-Be）するか検討・改善することが可能である。よって各情報システム開発者が、より大規模で複雑になる情報システムを、より堅牢な情報システムとして開発することを可能する。その結果、情報システムは社会インフラとして信頼性の高いサービスを提供し続けることも可能となる。

以降では、このようなソリューション・フレームワークを用いて今後の情報システム業界が取るべき方策の検討を行っていく。

10. 将来に向けたソリューション検討

◆ 外部環境の整理とソリューション検討

本章では、情報システム産業を取り巻く状況を整理し、それが将来どのようなシナリオが描けるのか、そし現在よりも正しく発展していくにはどのようにしていくべきなのか考えることとする。

まず情報システム産業を取り巻く状況を SEPTember 分析（Social（社会）、Economy（経済）、Political（政治）、Technology（技術））という観点で世の中の流れを捉えてみることとする。

① シナリオ作成方法

実際に SEPT 分析を行う前に、本章で行うシナリオを用いた検証方法について説明する。はじめにシナリオの作成方法について述べる。

角和昌浩（2005）によるとシナリオの作成方法には大きく２種類あり、規範的シナリオと呼ばれるフランス流のやり方と探索的シナリオと呼ばれる米国流のやり方がある。さらに探索的シナリオには、帰納的アプローチと演算的アプローチの２つのやり方がある。

２つの流派、つまり規範的シナリオと探索的シナリオの違いについて述べる。規範的シナリオはあるべき姿を思い描き、それに到達するまでのシナリオを示すものである。一方、探索的シナリオは外部の環境の変化を想定して、起こりえる複数の未来を描き出している。探索的シナリオを用いた場合、そのシナリオの利用者は、出来上がったシナリオを元に、次に取るべき行動を考えることになる。

また、探索的シナリオの２つのアプローチについてどのような違いがあるかというと、帰納的アプローチとは将来想定される事象を任意に洗い出し、それを適宜並べ替えながら論理的に矛盾のないシナリオにつなぎ合わせていくやり方である。また、演算的アプローチとは想定される未来に関わるデータを俯瞰して、未来を動かす“仕組み”または“構造”を見つけ出す。その構造を元にしてシナリオを作成するやり方である。

どのようなやり方を採るとしてもシナリオを作成するには、まず将来も確からしいと考えられる事象を考え、それをシナリオの基本とする。そして、次に未来を想像したときに不確かそうな事象を上げ、特にその中で重要と考えられる事象を抽出する。その重要で不確かな事象が起きるケースを考えることで、基本シナリオをその想定したケース分、派生したシナリオを作ることができる。

では、ここで実際に情報サービス産業を取り巻く環境を SEPT 分析により概観し、その後将来も確からしいと考える事象を抽出してみることとする。

② SEPT 分析

◆ 社会

はじめに述べたように、現代社会に情報システムは欠かせないものとなってきている。日常生活においてもその依存度は年々高くなってきている。コンビニの POS システムや JR の SUICA、また銀行の ATM も同様である。そのように日々の生活の中で多くの人々が知らず知らずのうちに利用している情報システムであるため、トラブルを起こしたときの社会的な影響も非常に大きく、万一事故が発生したときは新聞やその他メディアのニュースなどでも昔に比べて非常に大きく取り上げられるようになってきた。

実際我々が日常で利用する各種製品の構造にも大きな変化がみられる。主にハードウェアの組合せで製品を

実現していたが、現在ではソフトウェアとハードウェアを組合せることで製品機能を実現するようになってきた。特に最近ではハードウェアよりもソフトウェアの開発費用の方が大きくなってきているという事象も多くみられるようになってきた。このように、個々の製品自体に組み込まれるソフトウェアも非常に大きなものとなり、ソフトウェアに対する依存度が高くなっていることが分かる。

その結果、IT 産業に関わる労働者人口も年々増え続けているが新 3K（きつい、帰れない、給料が安い）と呼ばれる労働環境から、最近は IT 産業への就職を敬遠されるようになってきている。ソフトウェアの開発量が大体労働人口に比例することを考えると将来的に日本の IT 産業に従事する労働人口の変化は、産業の発展に大きく影響する要素であると言える。

◆　経済

情報サービス産業は 12 兆円程度の売上げのある産産業であり、食品業界や鉄鋼業と並ぶ売上高を持つ産業に成長してきている。“楽天“に代表されるようなインターネット企業もその動向が注目されるなど、情報サービス産業が経済界においても非常に重要度を増していると言える。

その背景として、“社会”で述べたように情報システムによるサービスの増加、または車や携帯などに代表されるように製品自体に組み込まれたソフトウェアの増加、またそのソフトウェア開発量の増加などもひとつの要因である。

しかし、対外的な問題として日本のソフトウェア製品は完全に輸出規模が輸入規模に比べて非常に小さく、その競争力強化は大きな課題である。また、近年ではその開発量の増大に伴い、各社がその開発費増大が問題となっている。そのため中国、インドなどに開発を委託する“オフショア開発”が本格的に動き出している。大手のシステム開発会社が買収などにより、中国、インドに関連会社を持つようになり、技術者のもつ IT スキルの高さと低コストを武器に全体としてコスト低減を図っている。

それ以外にも、NTT データや富士通は欧米の企業を買収することにより積極的に海外案件の受注などを目指した努力も行っており、例えば 2009 年 6 月 26 日の日本経済新聞（朝刊）には富士通が米国で４００億円のアウトソーシングを受注したことが報道されていた。同記事によると IT サービス事業の売上高世界ランキングでは富士通はアクセンチュアに次ぐ 4 位にポジションニングしており、日本企業の今後の動向は注目に値する。

IT 投資状況の傾向として設備投資額の 7 %程度が IT 投資に配分されるケースが大半である。しかし昨年度からの急激な経済状況の悪化により、今後の IT 投資傾向がどのような変化を見せるか注意が必要である。特に、最近では金融業界で起きていた銀行の統合などによる大型案件開発（つまり、IT 投資）が終わり、また NTT の NGN 関連の投資もあるが金融業界ほどの大型案件は生み出していない状況で､製品メーカなどでは新規に販売するライセンス費が中々伸びない状況にある。既に販売済のライセンスから得られる保守費用が新規ライセンス販売による金額を上回っている傾向にある。

◆　政治

日本のソフトウェア産業政策の経緯の振り返りを行う。1990 年代から始まり、 e-JAPAN 戦略が制定されたのは 2000 年になってからである。国家戦略として、ソフトウェア開発力の強化というソフトウェアの開発作業そのものを問題視してテーマとしたのは 2008 年の重点計画案が初めてである。それまでは個々の要素技

術の開発がテーマとなり、主な狙いは IT の利用者の利便性向上にあった。

2009 年には「高度情報化社会における情報システム・ソフトウェアの信頼性及びセキュリティに関する研究会の中間報告書」として、情報システムの品質向上に向けた取り組みを全面に押し出した報告が経済産業省からなされ、今後その取り組みの効果が期待されるところである。その中間報告の中では、契約に関わる法整備や人材育成についても提言がなされ、それまでソフトウェア工学中心に論じられ、あまり主題とならなかった課題についても問題として認識されるようになった。今後どのような方向性に議論が進むか注視する必要がある。

また最近では総務省を中心として“クラウドコンピューティング”に関する研究会を立ち上げ、日本での普及啓蒙を促進しようという取組が始まった。

また電子政府化などを起因にしてセキュリティに関する取り組みも政府中心に行っており、最近では情報セキュリティ政策会議が、「省庁のサーバー約 2900 台を 2013 年度末までに半減する」という発表を行った。サーバーの集約によりシステム運用費の削減だけでなく、サーバー台数を減らすことで情報の管理を行い易くすることを狙っている。

商習慣の整備という観点では、法務省が民法上の委任・請負契約に関する監督官庁であり、厚労省が派遣法を、公正取引委員会が下請法・独禁法を、そして内閣府が中小企業、個人情報保護政策を扱っている。情報システム開発ではゼネコン的な企業間の繋がりを持ち、開発を行っていることが多いため、特に民法の委任・請負契約に対しては注意を要する。近年ではその企業間の関係が曖昧になっているケースも多く、契約で定められた作業指示形態が取られていないために新聞などで問題として報道されることもある。開発のやり方自体が悪いために法律で規制されたことが守れないのか、それとも法律があるべき開発のやり方に合っていないのか、今後しっかりと判断すべき課題であると考える。

最後に、権利という観点では、文化庁は著作権の問題を主に扱い、特許庁は特許法により情報技術（IT）の特許管理を行っている。今後オープンソース化が進み、著作権や特許の考え方もどのように変化するのか、その動向に注意が必要である。

◆　技術

ソフトウェア工学として様々な取り組みがなされており、ソースコードを設計書から自動生成する方法、テスト手法、品質管理手法、各種開発ツール、そして開発言語などの分野で様々な製品・理論が生み出されている。しかし、そのように技術が発展しても、開発の現場での混沌さは何等改善されず、その恩恵を上手く活用仕切れていない現実がある。

今後ソフトウェア工学の研究により開発された新しい技術を有効に使うことができるように工夫することが大切となってくる。

また、次に情報システム自体のシステム構成について振り替えてみる。企業の情報システムと言えば、汎用機主流の時代が長く続いた。汎用機の時代にはそのハードウェアの価格が非常に高く、ソフトウェアはその付加的な意味合いしかなく、また、その提供方法も基本的には汎用機ベンダが一括して請負、提供していた。汎用機であるため、現在のように様々な企業で開発された製品を組み合わせるということはなく、各企業が独自の仕様を持ち各社が独自にソフトウェアを構築していた。例えば COBOL という言語は IBM 仕様もあれば日立社仕様も存在する。このような流れが 1990 年ころから次第に変化がみられるようになり、“オープン化“を

キーワードに各社が低価格なサーバーを発売し、またそのサーバー上で動くパッケージソフトを提供するようになってきた。情報システムの開発時には、そのパッケージソフトを組み合わせることによりある程度の部分を作り上げ、足りない部分を独自に設計してソースコードを作り上げていくようなやり方になった。しかし、最近では、その構成要素が多くなりすぎ、特に一般的に言われているのはハードウェア、つまりサーバーが多くなりすぎており、管理が煩雑になっている。またひとつのサーバーも CPU を使いきっているわけではなく、その余剰の処理能力を有効に使えないか、という課題がトピックとして持ちあがってきている。その解としてクラウド化という技術が注目されている。

現在多くの企業がオープン化によって分散したハードウェアを集約してコスト低減に結びつけようと、その集約手段としてクラウド化を検討し出している。

しかし、ひとつ問題になるのは、クラウド化するときには仮想化技術を用いることになるが、その技術を用いると、以前の分散したシステムの環境以上にシステムの管理が難しくなるということに注意しなければならない。ハード、OS、ミドルウェア、アプリケーション、という層構造の中に、更に仮想化層とも言えるようなものが介在することになる。それにより、今まではさきほど記述したようにハード、OS、ミドルウェア、アプリケーションという４層の管理で済んでいたものが、それだけでは済まなくなってしまう。仮想化技術の詳細を述べることは省略するが、ハードウェアとアプリケーションの連携が仮想化層によって複数パターン存在することになり、トレースしきれないのである。よって、もしハードウェアが故障した場合に、どのアプリケーションに影響が及ぶのか、現在の技術では迅速に判断することは難しいと考える。

結果として、想定されることは、汎用機時代のように複数社の製品を組み合わせて利用することから、複数製品を組み合わせる場合でもある１社に限定した製品になり、例えば上記のような問題も解決策を用意したひとつのスイート製品として提供されたものを利用することになるのではないだろうか。

または、例えば現在２つの製品で実現している機能が１つの製品として提供されるようになり、管理すべきパッケージソフトの個数も減少してくるかもしれない。飛躍しすぎかもしれないが、Google chrome も類似した考え方をもった OS かもしれない。多くのユーザが今後は Web 中心であり、全てがブラウザを通したユーザインタフェースになることを想定したアーキテクチャを前提とした考え方を前提にした未来では正しい方向性を示しているように考える。

③　シナリオ・プランニング

以上の内容を踏まえると、将来確からしい事象として

- 情報技術（IT）は 2020 年になっても存在し、発展している。
- 情報技術（IT）は今後さらに社会インフラとして重要度を増してくる。

という２つのことが言えると考えた。

また、将来に最も不確かで且つ影響を与えると考える事象は

- ソフトウェア開発費用の削減、品質向上に向けた技術はどのように発達するのか。
- 社会インフラ化した情報システムは、それに準ずる品質を満たすことができるのか。

という２つであると考えた。

これらのことから、下図に示すように４つのシナリオを想定した。

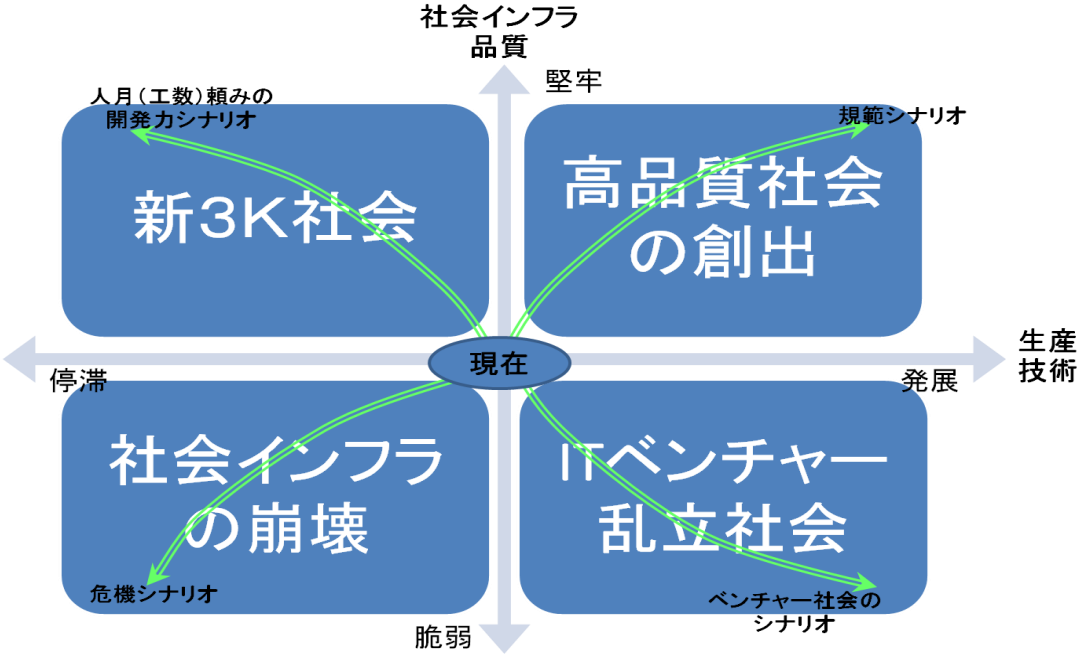


図 78　情報サービス産業の今後 10 年をシナリオ・プランニング

（角和昌浩（2005）を参考に作成）

それぞれのシナリオについて以下で説明を行う。

人月（工数）頼みの開発力シナリオ：

新たな技術開発に対する投資額が更に増加し、その行き過ぎた投資に抑止がかかり出す。その影響で、情報技術（IT）のレベルが衰退し開発力を向上させるような技術開発も進まない。結果として、人月（工数）つまり人手頼みの開発を行い、また社会システムとしての品質も開発担当者の経験と勘に頼るしかない状況になってしまう。その時にはアジアだけでなく低コストで且つ高い IT 教育水準の労働力を持つ国を世界各国から発掘し、協力関係を築くことになるであろう。

この場合、前述のマトリックス分析で言うのであれば人月ばかりが増えることになり、team と activity の関係がより複雑性を増していくことと同意である。結果として activity と artifact の関係、も一意性がなくなり、開発作業としては混沌とした現場になることが予想される。よって、マトリックス関係から導き出されるシステム品質は低下することは明白であり、その品質低下を抑止するための方策検討は必須である。

例えば、component と function の関係性を単純化するためにソフトウェアのアーキテクチャは予め規定しておくこと、などもシステムを構成する要素の関係性全てを混沌とした状況に陥らせないためにも重要な対策のひとつに成りえるものである。それにより function と component の関係は一意に決まることになるので、たとえ上位の team や activity の関係が混沌としても成果物である function や component に対して枠を設けておくことは混沌とした状況を軽減する効果は期待できる。その他の対策としては、多くの作業者をひとつのプロジェクトとしてコントロールするのではなく、小さなシステムに分割することで team と activity、activity と artifact の関係性をシンプルに保つことも一案である。つまり、マトリックスとして例えば 10×10 の行列ではなく、ひとつのプロジェクトを 2×2 の行列のような小さな次数のマトリックスで関係性を整理できるレベルに留めることで、冗長さをなくし難易度を下げる効果が期待できる。このような対策を打つことで規範シ

ナリオ、つまり高品質社会の創出という象限へ移行することが可能になる。

以上まとめると、マトリックス関係上では team／activity／artifact の複雑化が想定される。その対策として、component-function レベルの関係性予めを決めておくことにより、全部で７つあるマトリックスの中でも注意すべきマトリックスを限定していくことが可能になる。また、10×10 の行列ではなく、ひとつのプロジェクトを 2×2 の行列のような小さな次数のマトリックスを５つ考えて 10×10 のマトリックスを作るという手法も制御性を高める手法として効果的である。そのためには上記のように component 単位に team を決めてしまうとか、component 単位に activity を分けてしまうような対策が必要になる。

ベンチャー社会のシナリオ：

様々なアイディアで技術の発展が進むが、ベンチャー企業の域から出ることができず、その技術が永続的に維持・発展するまでに至らない。それは、例えば万一故障が起きた場合にそのベンチャー企業が存続しているかという現実もあり、社会インフラのように何年にも渡って永続的な利用にはリスクが大きいことに起因する。様々な技術が生まれ、デファクトスタンダードを巡る競争などにより、産業として活性化されることは間違いないが、結局スタンダード化されないまま終わってしまう。そのように永続性のない技術は社会インフラの品質向上には寄与せず、逆に脆弱なものにしてしまう。

ベンチャー企業のアイディアの多くはネットワーク分析で言うならばノードの難易度を低減させるもの、または一部のノードとリンクの関係性をブラックボックス化し、外部からはその複雑性を隠蔽化することでシステム全体からは一部を単純化してしまうものがある。情報システム産業全体をよりいい方向へ牽引していくものではあるが、その技術が標準化されることは非常にごく稀であり、多くの場合は類似技術の乱立・競争で終わってしまう。つまり、マトリックス分析で言うならば、ベンチャー技術に伴って少なからず品質向上などが期待できる。しかしマトリックスを構成する成分がベンチャー企業の技術の場合は時間をパラメーターとして持っており、その成分値が時間軸で変化し、難易度・複雑性を上げる要因になってしまうのである。結果として、構成要素の関係性を示すマトリックスのロバスト性を低下させてしまう。もし、標準化される技術であれば、標準化されるに伴って技術者にも普及し、その技術を扱うことに対する障壁が下がること、またその技術そのものの問題も改善されていくこと、これらの要因によって逆に時間軸の変化によって難易度・複雑性は軽減していく。そのように新しく生み出される技術を評価し、それを標準化することが重要であり、そのことによって規範シナリオに移行することが可能となる。

以上をまとめると、マトリックス関係上では artifact／component／function の難易度、複雑性が上がってしまうことが想定される。その対策としては技術の標準化が有用であり、技術が標準化されれば team や activity も何度か繰り返し利用する度にスキルが向上し、作業の質も改善されていく。それによって、開発されるシステムの品質も向上していくという好循環が生まれる。

危機シナリオ：

新３Ｋと言われ、労働人口の減少に伴い開発量も低下し、また技術力の低下にもつながる。結果、新たな技術革新も起き難い世の中を想定した。また、労働人口の減少は技術の“伝承“にも大きな障壁となり、品質を維持していくことすら難しい時代が来るかもしれない。現在のように工数に頼った開発、また開発の自動化を何年も試みているが上手くいかないこと、大手のシステムインテグレーターは実質的にその開発作業にはほと

んど関わっておらずその管理作業にのみ終始して開発作業全体を大きく変換したいというモチベーションが低いことなどから、この危機シナリオの実現度は低くはないと考える。

しかし、そのようにこのシナリオは現在の情報システム産業が成り行き任せで今後も進んでいった場合を想定してはいるが、このようなシナリオになることは絶対避ける必要がある。よって、全てを改善して規範シナリオ、つまり高品質社会の創出に向けて動くべきではあるが、少なくとも新３Ｋ社会または IT ベンチャー乱立社会への移行を目指し、そこから規範シナリオへのステップアップを図るべきである。

つまり、行列が全て全行列化し関係性が複雑化する方向になり、そして行列の各成分もすべて難易度が上がってく、当然行列自体の次数も上がり冗長性が増していくことになるため、その難易度・複雑性の上昇を抑える方策を取る必要がある。そのやり方として一足飛びに規範シナリオに向かうのも良いが、新３Ｋ社会または IT ベンチャー乱立社会への移行を踏んでから、再度規範シナリオに向かうやり方もあるということである。

以上をまとめると、マトリックス関係上では全ての要素が複雑化し、難易度も上り制御不能な状況に陥ることが想定される。よって、どの要素かを改善することを目指し、つまり step1 として新３Ｋ社会または IT ベンチャー乱立社会に移行し、次に高品質社会の創出に向けて移行することを考えてゆくべきである。

規範シナリオ：

情報技術（IT）進歩に伴い、その開発技術自体も進歩している。例えばソースコードの自動生成技術やテストの自動化技術などが進んでいる可能性がある。そのようにソフトウェアの品質だけでなく、生産性向上にも寄与する技術の発展により、日本全体の労働人口が減少していく状態であっても、社会インフラとして安定して稼働していけるだけの堅牢なシステムを開発・維持することが可能となる。それにより高品質な社会を実現しているであろう。

言い換えれば、このシナリオでは先ほどまでの３つのシナリオで指摘した改善点すべてを実行した場合を想定している。これらの改善を行うことを提案し、以降ではその具体策についてさらに検討を深める。

① アーキテクチャなど、規定することが可能な箇所などは明確に規定して、対角行列化すること（要素間の関係性を単純化すること）
② 小さなシステムに分割することにより行列の次数を小さくし、冗長性・干渉性を極力排すること
③ 時間変化による開発システム全体の品質変化を防止するために標準化を行うこと
④ ステップ論（危機→新 3K 社会→規範、危機→IT ベンチャー→規範）を考えて変革していくこと

これらを適宜取り込み、規範シナリオを描くために必要な改善策の実行方法について３つ観点（組織・プロセス・アウトプット）を基本にして検討を行う。

＜アウトプットの観点から＞

現在非常に多く用いられているアイディアはこの観点で利用されている。それは例えば、SAP のパッケージ製品であり、または Struts といったフレームワークである。これらの製品類によって、提供する function がどの component に実装されているか予め決まっており、それに準じて設計書を仕上げればよい。このように function-component-artifact が一意に決めることができる。これにより要素間の関係性は単純化することが可

能である。しかしながら、上流設計者は実装技術に弱いことが多く、どのような部品（component）によって要求機能が実装されるか理解しないまま設計作業を進めることが多い。そのため中間生成物も多く無駄な作業が生じているケースもある。対お客様説明向けに仕様書を独自に、つまり実装に向けた作業としては不要だが、作るケースもあり、しかもその仕様書が納品物である場合は本来不要なものでも情報システムが稼働している間は維持管理を行う必要が出てきてしまう。

このような実情を解決するには現在の請負・委任の契約についても見直す必要がある。上流工程または下流工程を担当する技術者のそれぞれの作業分担を適切する方策を導き出すために、海外での実情なども参考にしていく。

＜組織の観点から＞

前述のとおりゼネコン体制で多くの大規模開発を行っている。そのため多くの企業が複雑な関係を保ちながら作業を行っている。しかし、課題として指摘したとおり、一社では対応仕切れないため協力会社に多くの作業を依頼しなければならないのが実情である。しかしその結果、様々な弊害が起きている。これをマトリックス分析で考えてみると、team と activity の関係が対角行列で表されるものではなく全行列で表されるような関係性を持っているケースが多い。または team-team 間の関係をコントロールするだけの役割になっている team の存在があることが分かる。

前者の全行列の関係性を持っている場合、このような冗長性のある関係性を team-activity 間で持っているにも関わらず、その状態で下請け企業に作業を投げてしまうことは、その作業品質を管理していくことを考えると非常にマネジメントの難易度は高くなる。そのため、全行列で表されるようなところではなく、対角行列で表せる activity を見極めて協力会社に依頼することが重要となる。これにより協力会社も基本的には独力で作業を進めることが可能になり、マネジメントする側もその作業状況の管理は容易になる。他の関係者との調整事項が多くなるような全行列で表せる関係性をもって作業を進めることは下請け構造の下層にいくほど難易度は上がる。

また、プライムコンストラクターが team-team 間のコントロールに特化した役割しか持っていない場合も team-team 間のマトリックスが極力依存関係のないように、つまり対角化された行列で表せるように役割分担を決めることが重要である。team-team 間の依存関係が複雑になれば、当然プライムコンストラクターはコントロール仕切れなくなる。協力企業間で依存関係を生じないようにし、あくまでもコントローラーであるプライムコンストラクターと協力会社にしか依存関係がないようにすることに気をつけなければならない。情報のハブは複雑な中にしか生じない。そして単純な仕組みの中にはハブは生まれない。複雑性を抑えることがハブとなることなのである。

このような状況では、前述した通り component／function 単位に team／activity を分割してしまうことが小さな単位にマトリックス化することを可能にし（全体としては巨大マトリックス化していても）、小さなマトリックス内に閉じた冗長性・干渉性があるのみで、その小さなマトリックス間の冗長性や干渉性を低減することが可能になる。それによって複雑性を低減し、組織活動としての品質を確保する。

＜プロセスの観点から＞

この観点からでも activity-artifact の行列関係を極力対角化することを狙う。しかし、ソフトウェアでは、

開発当初からどのように部品構成で要求条件を実現するか決められないケースが多い。そのため、部品に対する設計書自体もどのような単位で作るべきか曖昧なまま開発作業を行っていき、その作業の中で設計書の単位を決めていくことが多い。最近では前述の Struts と呼ばれるフレームワークなどの利用により、機能構成、部品構成の大枠を当初から決められるケースもあるが、例えばデータベースアクセス部品はどのようなアクセスパターンを幾つ用意すれば良いか、などは開発当初から明確にできるケースはまだまだ少ない。そのようなことから元々ソフトウェア開発の方法論では WBS : Work Breakdown Structure のみに着目した考え方が主流であり、PBS : Product Breakdown Structure への意識が薄い。結果的に Product つまり作る部品そのものが曖昧であるため見積もりの精度も大きな課題のひとつになっている。また、作るべき部品が明確でないために冗長な設計書を上流工程（部品構成が明確になるまで）で多くの稼働をかけてまとめていることも少なくない。

しかし、今後は WBS : Work Breakdown Structure と PBS : Product Breakdown Structure を明確に意識することで冗長な設計を排除し、言ってみれば activity-artifact の関係を対角化していく必要がある。日揮プロジェクトサービス顧問高橋良之氏によれば、プラント開発などのような大規模プロジェクトでは縦軸に作業工程、横軸には開発する設備を設定したマトリクスをプロジェクト管理の基本に用いている。

	搬入・出荷設備			用役設備		製造設備	環境設備	建物・付帯施設
	搬出設備	貯蔵設備	搬入設備	空気供給設備	電気供給設備			
概念設計								
基本設計								
詳細設計								
構造設計								
製作・工事設計								
製作・組立								
建設工事								

図 79　WBS-PBS マトリックス

ソフトウェア開発においてはプロジェクト開始当初から横軸に、開発する部品やシステムの構成要素を列挙することは難しい印象があるが、プラント開発の場合は（そこで利用される情報システムの開発は除いて）、ハードウェアで構成されたものであり、準備されるハードウェアが持つ機能は予め明らかであるため、機能と部品の関係性を示すマトリクスはソフトウェアの開発のように、開発の度に変更されるものではないものと考えられる。その分、プロジェクトの複雑性は軽減されるが、プラント開発で扱う部品数が数万以上にも及ぶことを考慮すると差引きでは複雑性はソフトウェア開発と変わらないかもしれない。

どちらにしても、開発当初から最終的に実装する部品を意識して設計することで、極力小さな次数のマトリックス関係を作ることで冗長な設計を省くことができ、作業の単純化、つまり activity-artifact の関係を対角行列化することに繋がる。

結局は前述のとおり、アーキテクチャを規定し、標準化することで要素間の関連を単純化することで解決される。

＜情報システムの第 3 者評価の必要性＞

更に議論を発展させるが、このような方向性を堅持していくために企業内または国の法令としてもルール化できないものか考えてみる。

例えば、建築業界においては、都市計画法は国交省管轄で制定されているが、それに関連して建築基準法（国交省）、消防法（総務省）、自然公園法（環境省）、電波法（経産省）などがあり、またこれ以外に条例として各市町村レベルで制定された規定が存在する。建物を建設するにはこのような法規制その他を守ることが必要であるが、現状の情報システム構築にあたっては、そのシステム構造などについて特に規制は存在しない。将来的にもこのような無政府状態が良いのか議論が必要である。

はじめに述べたように、情報システムは社会インフラ化したものであり、その構造をあるひとつの企業が単独で検証し、稼働を開始する決断を下すだけで十分なのだろうか。情報システム開発においても建築業界に類似した仕組みを一部持ち込み、社会インフラとしての責任やその評価を明確にしていくべきではないかと考えた。現在のように情報システムの品質が第三者的に評価されることもなく、各社独自のやり方で作られることで社会基盤としての信頼性を担保できるとは考えられない。そのためにも今回提案したソリューションモデルによる評価などを用いることで改革を促したい。

・技術者レベルと作業の難易度の規制

マトリックス品質の値に応じて、それに携わるチームの技術レベルも規定する。例えば、PMBOK 取得者が最低〇〇名必要など、情報システム開発技術者のレベルに応じて携われる難易度のプロジェクトを規定することも検討すべき制度であると考える。

・安易な請負契約の規制

マトリックス品質の値によっては、仕組が複雑すぎるために請負作業は不可とし、委任またはそれに準ずる契約により協力会社との共同作業を許可するといった制度を考えている。米国のように T&M 契約により（委任契約に近い）、発注側と受注側も密に連携を取りながら開発することは複雑な連携を必要とする作業の場合には有用ではないかと考える。但し米国の場合は、作業を細分化して発注することで、能力の低い技術者の場合は直ぐに契約解除することでリスクヘッジしているようである。この点は日本も見習い、取り入れるべきポイントである。

・IT アーキテクトへの生産技術者的役割の定義付けと育成

各マトリックスをシンプル化していくことができる人材の育成、つまり、自動車などでいう生産技術者とは意味することが少々異なるが、全体のマトリックス関係をシンプル化する方策の検討を主な業務とする技術者の育成とその役割定義が必要である。

また、このような役割は既設のアーキテクトと類似している。現状のアーキテクトは function と component の関係を整理する能力はあるが、それ以外の関係性を見るまでのスキルはない。ここで言う生産技術とはアーキテクトが現在網羅している仕事と、それ以外のマトリックス関係を整理できる役割を指している。

・標準化（デファクトとデジュール）

“標準化”と“アーキテクチャ”をキーワードにして時間変化による開発システム（開発の仕組み）全体の品質変化を安定化させるためのソリューションを説明していく。

詳細を説明する前に、まずは“標準化”について基本的な説明を行う。標準化とは２つの種類が存在し、市場シェアの競争に勝った結果標準とみなされるケースをデファクトスタンダードと言い、コンソーシアムなどである技術に関係する企業その他が集まり合意形成した標準仕様をデジュールと言う。細かく言えば、デジュールにはコミュニティレベルで決める仕様と公的（国家・業界レベル）な

機関で決める仕様の２種類が存在する。

以下、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　中野冠教授の講義（システム標準化 2008 年度後期）より引用させて頂き、標準化のメリットを利用者の観点、製造者の観点、販売者の観点で述べる。利用者の観点では、陳腐化が遅くなる、選択の幅が広がる、価格の低下が見込める、安全・環境保全に役立つ、というメリットがある。また製造者のメリットとしては、コストの低下、市場の拡大、市場の形成・成長する景気を得られる、健全・公平な市場競争が促進される、技術情報が得やすい、ということが考えられる。そして最後に販売者のメリットとしては市場に障壁がなくなることで販売ターゲットが拡大するというメリットが得られる。

また留意事項として、標準化は競争領域に対して行うべきではない。なぜなら競争領域は変化が激しく標準化自体が追いつかないからである。非競争領域に対して、標準化を行うことで上記のメリットが得られる。逆にデメリットはどのようなものが考えられるであろうか。デジュールのデメリットとしてその標準が決まるまでに非常に長い時間を要することである。ひどい場合は策定した標準が結局は使われないケースもある。

しかし、そのようなことを考慮したとしても標準化により、市場が 1 企業にのみ独占されるものではなく幅広く公平なプレーヤーが活躍することが可能となるという状況を作り出すことは、国家レベルの視点から考えたときに産業を育成する観点で国の方向性を示すことにもなり、非常に重要なことである。

その標準化された市場の上で、さらに競争を促進し産業を発展させていく市場を作り出すことが重要なポイントと考える。

以上の内容を踏まえて、マトリックス関係として標準的な要素間の関係を作ることが必要でないかと考える。要素間の関係性は企業間の競争を必要としない概念レベルの問題であるので、標準パターンを構築すべきである。それにより前述の企業でのルール化または国としての法令化なども可能になる。その標準化されたパターンの中で、製品としてカバーされる箇所や製品として高度化される要素が出てくることで、その標準化された要素関係で成立しているシステム品質が向上していくのである。そのような発展により、情報サービス産業全体の安定した発展が期待でき、時間変化によって品質が揺らぐような仕組みを防止することが可能になると考える。

現実社会では行列成分または行列自体のノルムが時間に依存した関数であるため、品質が揺らぐ。初期はラーニングカーブによって品質が低く、慣れると品質が上がるが、数年経つとその技術が廃れて技術者も減少し、保守が困難になり全体品質が低下する。現在のような 2-3 年周期の技術進歩は技術者のスキルアップによる品質・生産性向上の効果を低下させる要因になっている。このような状況を改善していくという点からも、標準化すべきマトリックスやその成分を明確にしていくことは必要である。

そして、最後に今後全ての業界でも問題となると考えられる労働人口の減少について考えてみたい。規範シナリオでは生産技術も発展し、労働人口は少なくても開発量はさほど減少することがないかもしれないが、永続的な発展を考えたときに労働人口の減少は致命的な問題となる。

よって、現状のオフショア開発の流れやその他の政治経済の状況から考えて、アジア圏での協業体制の確立

がソリューションとなるのではないかと考える。アジア圏との協業に対して、提案したモデルが有効化であろうか。

＜アジア各国との協調開発方法論＞

以降で、アジア各国とのオフショア開発とその後の有効な関係性を築いていくにはどうすべきか私案を述べる。「アーキテクチャ設計を日本で行い、基盤部分は日本で製造。その基盤上で動く各要素の開発はアジアで」という開発形体を提案したい。

アーキテクチャは設計者の思想そのものである。それにより機能をどのように分割してモジュール化するのか、そしてそのモジュール単位も規定されることになる。よって、そのようなシステムの根幹をなす部分は日本でアーキテクチャ設計を行う。

今回の提案モデルでは標準ソフトがそのアーキテクチャを決定付けるものとなっている。その基盤（アーキテクチャ）で規定された各要素（分割機能）を個々にアジアで設計、製造する、つまりモジュール単体の開発をアジアで行うような仕組みが理想的である。

システム全体のアーキテクチャはあくまでも日本内部で留保しておくことで、根幹的なノウハウが流出することはない。各要素を上手く分担してアジア各国に振ることで、そのシステムの全体像明らかにしコピーするためにリバース生成してしまうことも防止できると考える。

将来的にはアジア諸国で日本のシステム開発を行うことにより、アジア諸国の国内企業へ日本で設計したシステムを導入することを視野に入れていきたい。

システム仕様は企業の業務ノウハウそのものである。ということは、要素開発をある程度身につけた現地企業は、現地の他業界企業へその業務ノウハウを元にした提案ができる素養を身につけたとも言える。よって、そのようなオフショア企業には日本から出資し、しっかりとした資本関係を持った上で根幹となるアーキテクチャ設計内容も公開し、現地企業向けの受注開発を行っていくという将来的なシナリオを描けないだろうか。欧米発の既成製品との競合は当然発生すると想定するが、今後 10 年レベルで考えると、自国でソースコードレベルまで見ることができないようなパッケージ（既成製品）は廃れてくるのではないかと推測する。近年のライセンス商売の状況は次第に雲行きが怪しくなってきており、新規のライセンス販売額より保守費のほうが大きくなってきている。また OSS（オープンソース）というキーワードも忘れてはならないだろう。
それは、内部で何をされているかわからない製品は故障時の対応で困ることはもちろん、国家や企業のセキュリティ上も懸念されるのは当たり前ではないか。中国がソフトウェア製品企業に対しソースコードの公開を要求した考え方も理解できなくはない。

そのようなことから日本企業とアジア各国の協調したソフトウェアシステムの開発体制を築いてはいけないだろうか。

アジア各国で日本と同じアーキテクチャ上でいろいろなソフトウェアが動くことになる。それは、システム間の情報のやり取りを決める上でも重要なことであり、波及効果も大きい。現在の Saas や SOA などといったキーワードに匹敵するような影響力のある概念ではないか。

以上により、アジア協業に対しも有効なソリューションになっていると考える。

◆　内部環境の整理とソリューション検討

①　現状の課題と他業種の状況

現状の情報システム開発を行う組織体の課題について議論を深め、最後に To-Be モデルとして提案を行う。

＜ヒエラルキー型組織がヘテラルキー化する要因＞

組織の品質については、INSEAD 助教授 Manuel E.Sosa（2008）は「TIM（チーム・インタラクション・マトリクス）などを用いることにより、コミュニケーション」の上達を図ることを提案している。

構造化した指揮命令系統を基本としていると、ネットワーク化した現在のプロジェクト内で行われる情報のやり取りをマネジメントし切れずに、コミュニケーションロスも発生してしまうであろう。

ネットワーク化した現在では情報のハブが誰で、どのような構造で情報共有がなされているか抑えること、またそれにあったコミュニケーションシステムをデザインする必要がある。それは肩書きのみで描かれる体制図とは異なり、実質的なものでなければならない。

旧来の構造型組織の場合、例えば下図のように最下層の“e”はプロジェクトリーダー“i“に情報を伝達するために”h”を介する必要がある。当然そこで起こるコミュニケーションロスがあるため情報伝達の正確性確立を”p”とすると（p<1）、i に届く情報の精度は p^2 であり、さらに"a“と情報伝達を行うときには"g"を介するため、さらに精度が低くなり p^4 の精度となる。情報精度だけでなく情報伝達速度も同様に悪くなるため、当然バイパス経路を使って直接のコミュニケーションが行われるようになる。そのときの精度は p であるため、プロジェクトリーダーを通したやり取りよりも格段に情報精度が高く（$p > p^4$）、しかも情報伝達速度が速まる。

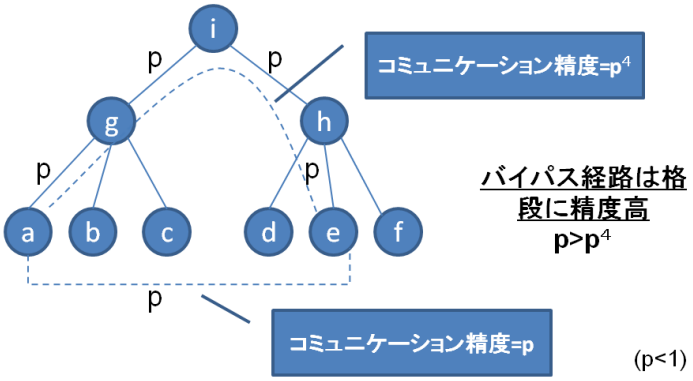


図 80　構造化組織のコミュニケーション精度

よって、大規模なプロジェクトになればなるほど、このようなバイパス経路が盛んになることは当然であるが、このバイパス経路が増加しすぎることはプロジェクトがカオス状態になることを意味する。しかし、以前は一社で少なくとも主従関係が成り立っている会社間で開発を行っていたことからヒエラルキー型で開発を実行してきたが、最近では参画するプレーヤーも増え且つ特に会社間に主従関係もなく対等な関係で参画するケースも多い。結果、上記の理屈で情報ルートとしてバイパス経路も発達してくることは抑えられず、多くのプロジェクトでは初めに示した統計データのように失敗プロジェクトに見られがちなカオスの世界へ陥っていくことになる。

それを避けるためにも、このバイパス経路を限定すること、そして、どのような経路が活性化しているのかということを把握することはプロジェクトマネジメントの重要な要素のひとつである。

＜アーキテクチャと組織＞

ソフトウェアの場合は前述のとおりハードウェアの製品とは異なり、開発すべき部品（システムを構成する要素）が開発当初は曖昧な状況であることが多い。そのため、何を誰が開発するのかその分担が漏れてしまうことや、また誰が何を開発しているのか曖昧であるため、本来各要素間の連携方法を確認しておくべきことを怠ってしまうこともある。結果的にそのような場合は、テスト段階などで問題が発覚し、手戻り作業として修正を行う必要が出てくる。

逆にアーキテクチャが明確であれば、どのような部品を作らなければならないか、そしてその部品を開発するときには、他に何の部品との連携を考慮して開発する必要があるのか明らかである。そして、その部品を開発する担当を漏れなく分担することで、担当者間のコミュニケーションロスも軽減され、例えばインタフェース仕様の調整漏れ、などといった問題が防止できる。INSEAD 助教授 Manuel E.Sosa（2008）によれば、エンジン開発プロジェクト（ハードウェアであるため、はじめからアーキテクチャは明確）のコミュニケーション状況と製品品質を分析するときに、次のようなステップで調査を行っている。

プロジェクトの担当名を縦横に並べたマトリックス表を用いて、アーキテクトにコミュニケーションするべき担当を示すマス目にチェックをしてもらった。そして次に各担当に自担当はどの担当とコミュニケーションする必要があるのかチェックしてもらった。そのチェックした箇所が完全にマッチングするならば設計者間のコミュニケーションは上手くいっており、情報共有がなされていることがわかる。しかし、チェック箇所がずれている箇所はマネジメント上、要注意なポイントとなる。何故ずれているのか理由を調査することで、重大な問題を事前に発見できるのである。

このようにシステムのアーキテクチャを明確にして、そのアーキテクチャ情報を把握しておくことはプロジェクト内でどの担当とどの担当とコミュニケーションが必要であるのかという情報ルートはもちろん、そこに潜んでいる潜在的な問題をも把握することができる。つまり、ネットワーク化したプロジェクトの情報をコントロールするハブ的な役割を担うことが可能なのである。

更に議論を発展させて述べるのであるならば、アーキテクチャをコントロールすることは各要素の機能分担もコントロールすることができることから、作業を他企業へ委託する場合でも委託した部品が他の部品とどのような関係性をもつのか理解できているため、所謂“丸投げ”状態にはならない。具体的にどのような要素でシステムは構成されているのか、そして、それがどのような関連性をもっているのか大枠を把握した上で協力会社へ作業を委託するならば、各委託先から報告される作業状況の把握、そこで上がってくる課題への適切な対処方法の判断などで誤ってしまうケースは少ない。しかし、そうではなくアーキテクチャ設計すら他社に委託する場合には、上記で説明した通り各社が何をやっているのか分からない状況で全てを判断することになる。それで正しい判断が出来るわけがない。そのように考えていくとアーキテクチャ設計情報を把握することは、ゼネコン体制での開発においても、上位企業は下位企業の作業品質のコントロールが可能とするための重要な手段である。

＜トヨタ自動車のアーキテクチャ指向組織＞

アーキテクチャを考慮した組織構成という観点でトヨタ社の開発組織が参考になる。情報システム開発においてもこのトヨタの開発組織を参考にすることを提案する。慶應義塾大学大学院佐々木正一教授（元トヨタ自動車株式会社）の講義、また服部秀雄（1991）によると、トヨタの開発システムでは、“チーフエンジニアシ

ステム”という開発体制をとっている。

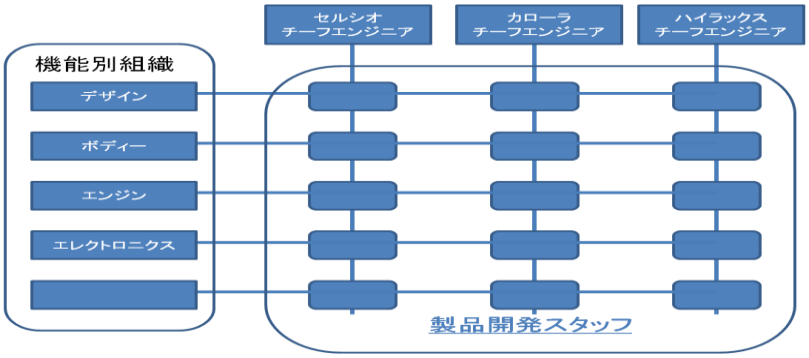


図 81　チーフエンジニアシステム概念図

「トヨタ自動車（株）における研究開発関連組織間の連携システム」

服部秀雄（1991）を参考に編集

横軸は開発する車種であり、縦軸に機能別組織、つまり車を構成（アーキテクチャを構成）する“要素別”の組織のことである。開発する各車種に各要素から担当者がアサインされ、それを専属のチーフエンジニアが取りまとめ、コーディネートしてひとつの車を開発するスタイルである。

今後電気自動車の発達・普及に応じて大きな変化が起こってくると考えられるが、現在の自動車ではどのメーカであろうとどの車種であろうと、そのアーキテクチャはほぼ同じである。車の開発を行う上ではアーキテクチャを構成する各要素が他のどの要素と関係があるかチーフエンジニアはその関係性を把握している。つまり INSEAD 助教授 Manuel E.Sosa の指摘していることを実践できているのがトヨタのチーフエンジニアであると考えられる。

マトリックスを活用した視点でさらに分析を進める。この場合トヨタの組織は Component と Team を１：１に結びつけていると診ることができる。そしてトヨタの場合は系列を活用した開発であるので、その component 単位に関連メーカへ作業を発注していると考えれば、各 component の開発プロセスはそれぞれ独立しており、activity や artifact も他の component との依存関係が非常に少ないモジュールとして開発をしていると考えられる。

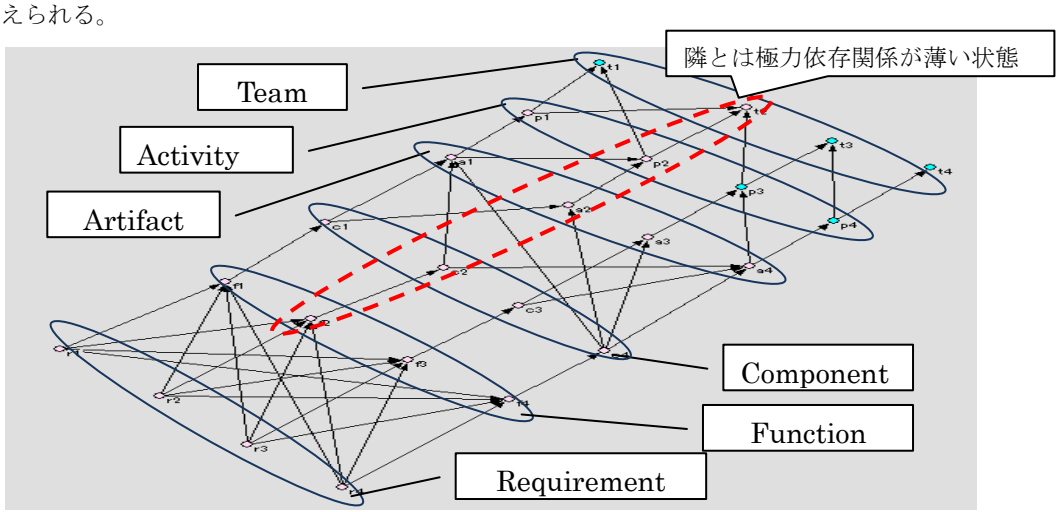


図 82　自動車開発の組織

関係性をネットワーク図でイメージとして表現したが、このように隣り合う要素との関係が複雑になること

を排除し、team から function までの関係性を作ろうとしている発想であること言える。更にこれをマトリックスで見てみると、個々の component 単位にクラスター化し、全体としては対角行列に近い構成を作っていると考えることができる。

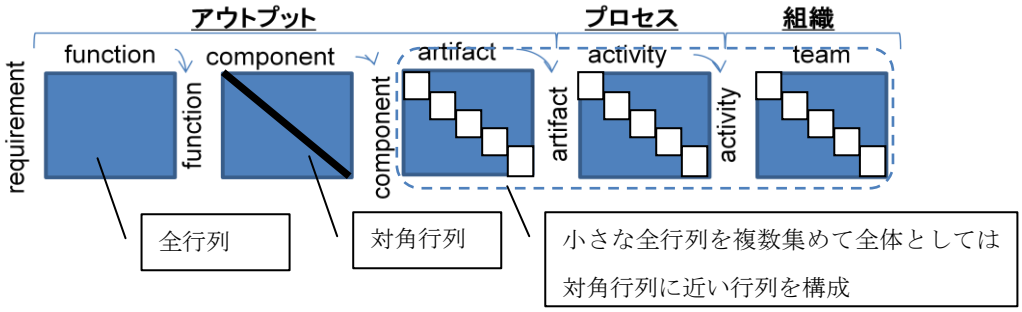


図 83　自動車開発組織の行列表現

さらに、上記では requirement から team までであるが、模式的に needs から team までの関係性を表現すると

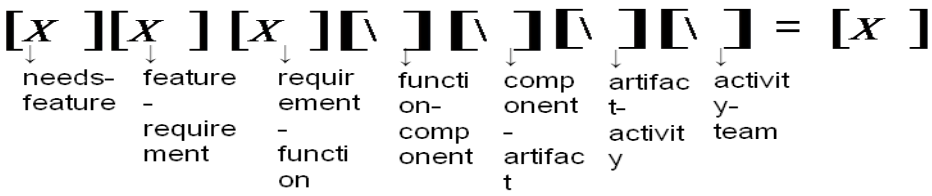


図 84　自動車開発組織の行列式

というように記述することができる。([x]=全行列、[＼]＝対角行列)

このように、全行列があるためにシステム全体として各行列を掛け合わせると全体も全行列になってしまうが、左から３つの行列のみに全行列を集めてしまうことでシステム全体のコントロール性を少しでも維持しようと考えた仕組としてみることができる。

function（機能）と component（部品）の関係は、細かく言えばひとつの component に対して複数の function を持っているケースもあると考えられるが、大凡対角化された関係である。そして最後に function と requirement の関係になるが、この関係は非常に複雑で全行列に近い。何故ならば、先ほども述べたように似た構造を持った各社の自動車ではあるが、完成した車自体、デザインは当然であるがその他それぞれの乗り心地も異なり、操作性も異なる。これは顧客からの要求を実際に function に落とし込む段階で、各社がその function に要求した機能仕様が異なることの結果であり、その落とし込み方はトヨタの場合であればチーフエンジニア次第であり、そのかじ取りに全責任を持ち実施している。

これまで見てきたことをまとめると、component から team（組織）の関係がおおよそ対角行列で現れされる関係、つまり１：１の関係になっているため、全体としては非常に単純化された組織モデルとなるが、唯一全行列化した関係は function と requirement のマトリックスである。このような関係性は、コスト、品質の観点からは全てがその単純化された構成になっていることから、各チームの能力がそのまま反映されて製品が出来上がるタイプになっている。3 万点もの部品を扱いまた多くの下請け企業を抱えている自動車産業にでは、このような関係性を保つことで、系列会社をコントロールしてコスト競争に競り勝ち、海外のメーカと比較しても高い品質を維持しているのである。そして、最終的に顧客に満足してもらう自動車を作るには

requirement と function の関係または、今回省略している feature と requirement、needs と feature の関係を上手くコントロールする必要がある。先ほども述べたように非常に多くの部品を扱い、非常に多くの系列メーカから成り立っている自動車産業をコントロールするために、極力対角化した関係を作り出そうとしてきたが、やはりそれだけの大規模なものを全てシンプルな対角行列化することは出来ず、requirement、feature、needs の関係性を自動車開発の最後のキーポイントとして考え、ここにチーフエンジニアに決定権を集約することで自動車開発という大規模なシステム全体のコントロールの難易度を低減させていると分析できる。

②　情報システム開発プロジェクトの留意点と組織の定量的議論

＜ソフトウェア開発を考える上での留意点＞

ハードウェア製品の開発の代表格である自動車の開発について分析を試みたが、そのノウハウをソフトウェア開発に生かしていく上で何が障害になるのか、どのような点に留意することが必要なのかここで考えてみる。

ハードウェア製品の開発においてはどのような製品を作るか要件定義後に物理設計に入る傾向が強いようだが、ソフトウェア製品の開発においては、要件定義後に論理設計を行い、その後物理設計を行う。論理設計とは、要件を満たすためにどのような機能が必要なのかを洗い出す段階である。その時洗い出した必要な機能を実際の物理的なものとして何によって実現するか、ということは考えない。その実現手段を考えるのは物理設計の段階である。

ハードウェア製品の場合は、要件により必要な実現手段、つまり必要な物理的な部品・製品がイメージし易いために要件定義が終了後に即物理設計に入ることが多いとようである。言い換えるとハードウェア製品はアーキテクチャが直感的にイメージとして湧きやすいのである。その結果開発当初から、ひとつのハードウェア製品を構成する要素が作業者の中でイメージし易く、そのためその組立工程をスケジュールも考えやすく、その作業工数を見積もることも可能である。

ハードウェアのようにアーキテクチャが当初から明確な場合、上記のとおりそれを構成する要素が明確になり、また要素毎に求める品質を予め規定することが可能である。そして構成要素が明らかであるから、その組立手順も決まってくる。そして、必要な構成要素や作業手順が明らかであるから、その作業に関わる技術者にどのようなスキルが必要なのかも明確にし易い。そのため、技術者も集め易くまた育成し易い。このようにアーキテクチャが明確なことによるメリットは大きい。

ソフトウェアの場合は、要件定義→論理設計→物理設計→実装（組立）という段取りになることから、ある要件を実現するのに必要な要素（製品・部品）が洗い出されるのは物理設計時になる。つまりその工程が完了するまで、正確な作業工数は分からないのである。

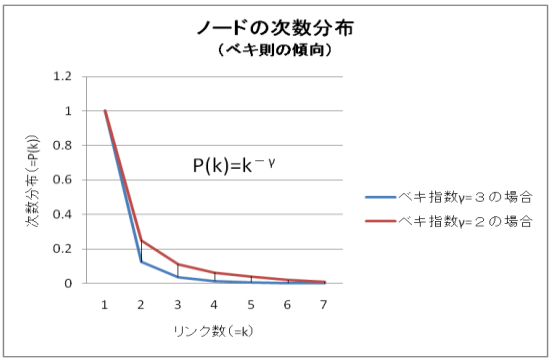
当然ソフトウェアの場合、何を作るかはじめからわからないため、どのような作業プロセスで進めるか、その具体的な内容は規定し難い。またどのような技術スキルが必要なのかも開発当初には分からないこともある。結果、前述した品質を導き出す算出式に代入して考えると、ソフトウェアの品質は予期し難いものになってしまうことが容易に理解できる。テストなどでその品質のチェックを行っているが、偶然上手く作られていたのか、しっかりとした作業を行った結果上手くテストをクリアしたのか、ではテスト工程での結果が同じでも大きな違いがある。

このようなことから、アーキテクチャが開発当初曖昧であるソフトウェアの品質を作り込むことはハードウェアとは異なった難しさを持っている。

＜複雑ネットワーク論による定量的分析＞

前述の通り INSEAD 助教授 Manuel E.Sosa（2008）によれば、エンジン開発プロジェクトのコミュニケーション状況と製品品質を分析により、システムのアーキテクチャを明確にして、そのアーキテクチャ情報を把握しておくことはプロジェクト内でどの担当とどの担当とコミュニケーションが必要であるのかという情報ルートはもちろん、そこに潜んでいる潜在的な問題をも把握することができる。つまり、ネットワーク化したプロジェクトの情報をコントロールするハブ的な役割、つまり、どのようなコミュニケーション経路が必要か限定し、その経路を活用することが可能なのである。

ここで、組織の規模とハブの関係について考察を加える。複雑ネットワークモデルの研究によると“スケールフリー性”という特性がある。その法則とは頂点の次数分布がベキ法則に従うというものであるが、それは例えば増田直樹・今野紀雄（2006）によると、1000 人の中で友人が 2 人いるのは、ベキ指数”3”の場合「1000×0.125（リンク数”2”の次数分布数）=125 人である。」という法則である。



リンク数＝k	ベキ指数γ=3の場合	ベキ指数γ=2の場合
1	1.000	1.000
2	0.125	0.250
3	0.037	0.111
4	0.016	0.063
5	0.008	0.040
7	0.003	0.020
10	0.001	0.010

図 85　スケールフリー性

増田直樹・今野紀雄（2006）より編集し作成

ここでは、大抵のケースでは“２～３“程度の数値を取るようであるため、ベキ指数を“２”の場合と“３”の場合を参考までに記述した。つまり、このことから 1000 人程度の規模のプロジェクトであるならば、10 人と情報伝達できるルートを持つ人は 1 人（γ=3）～１０人（γ=2）程度しかいないのである。このことから考えて、10 以上のチームに分けてもハブとなるプロジェクトメンバはフォロー仕切れない状況に陥るともいえる。20 人と情報ルートを持つ人は上記表には記載していないが、γ=3 では 0 人、γ=2 では 3 人である。これはγ=3 ではこのプロジェクトをマネジメントできる人がいないことを意味する。つまり、あるチームがプロジェクトのマネジメント対象から外れてしまう状況であり、大きなコミュニケーションロスによって、品質や生産性の低下を招く原因になると推測する。よって、プロジェクト内の大きなチーム構成は 10 以下にする必要がある。

また仮にプロジェクト内でのチーム数が 10 であった場合、そのチーム間が協調して行う作業は多くても 3 チームで行う作業が最大であり、4 チームでは協調性が低下してくることが予測される。つまり、10 チーム中で 3 チームと関連が持てるチームは上記表（リンク数=3）よりγ=3 では 0.3、γ=2 では 1.1 であり、4 チームとするとγの値に関わらず両方とも“１“以下の数値しかとらないのである。アーキテクチャ設計を行う

場合、各要素の関係を考えるときでも要素間の関係は３以下に抑えるような設計を行っておくことが、予想外のバイパス経路を防止する対策にもなり、情報流通を把握しやすく、要素単位に設計チームが分担するときに効果を発揮することになる。

また、もうひとつ複雑系ネットワークに共通の性質がある。それが“スモールワールド性”である。ネットワークの大きさ（参加者数）に比べて、互いの平均距離は小さく且つクラスター性がある、というのがその性質である。世界中の誰でも平均６人程度を介せば、皆につながる。という説明がされることも多い。数学的には「互いの平均距離は自然対数“ln（ノード数=ネットワークへの参加者数）”に比例する。」という性質がある。この性質を用いて、ひとつのチームに属するメンバ数について検討を行う。

ノード数	ln(ノード数)	平均ノード間距離	比例係数
1	0.000	0.0	1.5
2	0.301	0.5	1.5
3	0.477	0.7	1.5
4	0.602	0.9	1.5
5	0.699	1.0	1.5
6	0.778	1.2	1.5
7	0.845	1.3	1.5
10	1.000	1.5	1.5
15	1.176	1.8	1.5
20	1.301	2.0	1.5
30	1.477	2.2	1.5
50	1.699	2.5	1.5
100	2.000	3.0	1.5
300	2.477	3.7	1.5
500	2.699	4.0	1.5
1000	3.000	4.5	1.5

図 86　スモールワールド性

一般的に言われているようにひとつのチームは６人前後が適切である、という論理から考えて、互いの平均距離が“１“前後になるように比例係数を 1.5 に設定して上記表を作成した。６，７人程度までのチームであれば、誰がリーダーに成らなくても互いの平均距離が１程度、つまり直接互いに知っている関係であるため、その自然なコミュニケーションにより情報共有が可能である。しかし、１０名以上になると次第に平均距離が２に近づき、誰かを介さないとある２人のコミュニケーションが成立しないケースがあることを示している。１０名程度のプロジェクトであれば、目的意識さえ合っていれば自然と仕事は流れていくが、それ以上であれば何かしらのコミュニケーション・ルールが必要であり、それをコントロールするための仕組みが必要ということである。言い換えればアーキテクチャ設計を１０名以下のプロジェクトではアーキテクチャ設計の品質が悪く、要素分解が上手く出来ていなくても開発作業は進められるが、それ以上の規模では注意が必要ということになる。また、１つのチームの最大が１０名とするとスケールフリー性の「ひとつのプロジェクトは１０チーム以下が望ましい」という論理と合わせると最大１０人×１０チーム＝１００人規模のプロジェクトが適切であると言える。これ以上の規模のプロジェクトの場合は、サブシステム分割を行い、ひとつのサブシステム規模を１００人程度に抑え、各サブシステム間の関係も互いに”疎“な独立関係を保てるような設計を行うことが必要となる。

互いに疎な関係を保つには、独立性の高いアーキテクチャ設計に基づいた要素分解をすることにより、その要素ごとにチーム構成を整える必要があるとも言える。
このようにして無意味にバイパスルートが増加することを防ぎ、コントロール性を向上させることが重要である。

以上により、チーム構成を考える際の要員数を明らかにすることができたが、次にそのチームはどのような構成をとることが良いのか、またチーム以外にも作業プロセス、成果物などもどのような構成・関係性を持つような仕組みにすれば良いのか明らかにしていく。

◆　ソリューション　～IT 業界ではどうすべきか

①　外的環境分析と内的環境分析から

＜外的環境から＞

シナリオ・プランニングから得た各シナリオを概観すると、規範シナリオ以外の３つのシナリオでは function から team までのマトリックスに影響を及ぼすことが分かった。よって、極力対角化された行列関係に各要素を保つようにシステム設計をしておくことで、どのシナリオになってもロバスト性を維持していくことを考えた。

例えば、技術者が非常に増えたとしても、その作業分担が互いに干渉しないような役割に仕切っておくことで技術者層がより厚くなることはあっても、カオス状態に陥ることは防止できる。逆に層が厚くなることで技術者間では良い競争が生まれるかもしれない。

このように、function から team までを対角行列化する対策は先ほど紹介したトヨタの自動車開発事例に類似している。トヨタを参考にした対策を取ることで情報システムの開発も非常に高い品質を維持できてくるであろう。

needs から requirement までの関係性は、例え複雑化していても製品・サービス自体の品質値として反映されることがない。一般的に企業の品質保証部が管理する品質値の基準となるのは function が定義付けられる requirement のみであり、その requirement を基準に品質の良否を判定する。しかし、feature や needs は製品・サービスの品質値の基準にはなり得ず、企業側からはその管理範囲に入っていないのである。

また needs から requirement までの関係性は顧客要求、市場要求といった作り手となる企業にとっては、自社のコントロール外であり自社のコストを決める直接要因ではない。一方 function から team までは自社のコストを決める直接的な要素であり、そのコスト低減を図るためにはケーススタディーでもみたように互いに干渉しあわないような対角化された関係性が好ましい。互いの関係性を独立にしていれば、もしそれ以上のコスト低減を図ろうとするならば、行列の各成分値を調整すること、つまり個々の構成要素のコスト低減を図ることを行えば良い。しかし needs から requirement までが幾ら複雑な関係でも、それは販売価格で調整すれば済むのである。最終的に requirement－function の関係が対角化されていれば、効率的にそして高品質なもの作りが可能なのである。

このようなことから、品質・生産性の面からも function から team までを対角化し needs から requirement までは全行列化しておくことで、どのようなシナリオになったとしても十分に社会基盤として成立する情報システムを開発することが可能であると考える。

＜内的環境から＞

現在の大規模複雑な情報システム開発プロジェクトでは、大抵の場合下記に示すようなマトリックス構成になっているのではないかと考える。アーキテクチャに関する意識が低いため、PBS が抽出し難くアーキテクチャを構成する要素ごとに作業、成果物などを定めることができないような状態である。そして、要求条件と一言で言われることが多いが、顧客の抱える needs、その feature、requirement は複雑な関係性の上で定義されている。特に情報システムの要求仕様をまとめていく上では、独立公理として機能間の独立性を高めようと

いう意識は多少働いているように考えるが、第 2 の公理である情報公理、より少ない情報で機能設計を行える設計が良い、という考え方については殆ど意識されていないように考える。実際、顧客と要求条件を洗い出し整理する作業の中では、その整理するまでが精一杯であり、またどのような単位で機能を作るべきか考えながら作業を進めていることから、“より少ない情報で機能設計ができること“という条件を満たすことが困難なのである。その結果 requirement と function の関係も複雑になってしまう。実際、ある機能の設計書を書きだした段階で、要求条件書のページを色々捲って、その条件を整理し直すことは多くの設計者に経験があることではないだろうか。さらに function と component の関係性については、1 つの component にひとつの function が実装されることもあるが、ひとつの component に対して複数の function が実装されることも多い。よって、全行列にはならないが三角行列に近い構成になっていると考えられる。Component と artifact も同様にひとつの artifact に対して複数の component の仕様を記述することもあり、またひとつの component に関わる仕様が複数の artifact に跨っているケースもある。よって、全行列の関係になっていると言える。Artifact と activity の関係はある activity で複数の artifact が作成されることもあるし、またある artifact は複数の activity で更新されながら完成するものもある。よってこれも全行列の関係である。そして最後に activity と team の関係では、大抵の場合で team はある役割単位に構成されているのが基本設計レベルでは殆ど混在して activity を行っている。よって、team と activity の関係性を整理するならば全行列が一番近い。

$$[X]_{\text{needs-feature}}[X]_{\text{feature-requirement}}[X]_{\text{requirement-function}}[LT]_{\text{function-component}}[X]_{\text{component-artifact}}[X]_{\text{artifact-activity}}[X]_{\text{activity-team}} = [X]$$

（[x]=全行列、[LT]＝三角行列、[＼]＝対角行列）

図 87　現状の大規模情報システム開発の行列式

このように結果として、現状の大規模システム開発組織全体の特性は行列式の掛け算した結果になるので、当然全行列になる。7 つの要素の中でひとつも対角行列がないために非常にコントロールし難い状況にあることが理解できる。

このような行列関係にある組織体を対角行列に極力近づけてロバストな組織体に改善するためにどうすれば良いだろうか。先ほどトヨタの自動車開発組織を紹介するなかでアーキテクチャ指向という考え方について述べたが、以降で大規模情報システム開発プロジェクトの組織体について To-Be モデルを提案する。

②　To-Be モデル

ソフトウェア開発においても、アーキテクチャを標準化し、そのアーキテクチャを構成する要素単位に組織化することで、トヨタの開発体制に類似した組織を構築できるはずである。そして、近年 IPA（独立行政法人情報処理推進機構）の IT スキル標準で定められた IT アーキテクトの様な役割は、トヨタで言うチーフエンジニアであり、この IT アーキテクト的な人材の育成が急務なのである。但し、この IT アーキテクトは前述した生産技術者としての観点を役割として持つべきである。今までに挙げた 8 つの要素を極力対角化関係に整理するスキルを持った技術者が必要なのである。

現状では図に示すとおり、業務中心でプロジェクトを設計している。顧客の要求条件を基本設計の段階でシ

ステム化の観点でカテゴライズし直し、それをベースにして詳細設計を行っている。詳細設計の段階で初めてアーキテクチャを構成する各要素、つまり component を抽出するため、結果的には元の要求(requirement)に対して function までの関係性はメッシュ構造になってしまっているのである。

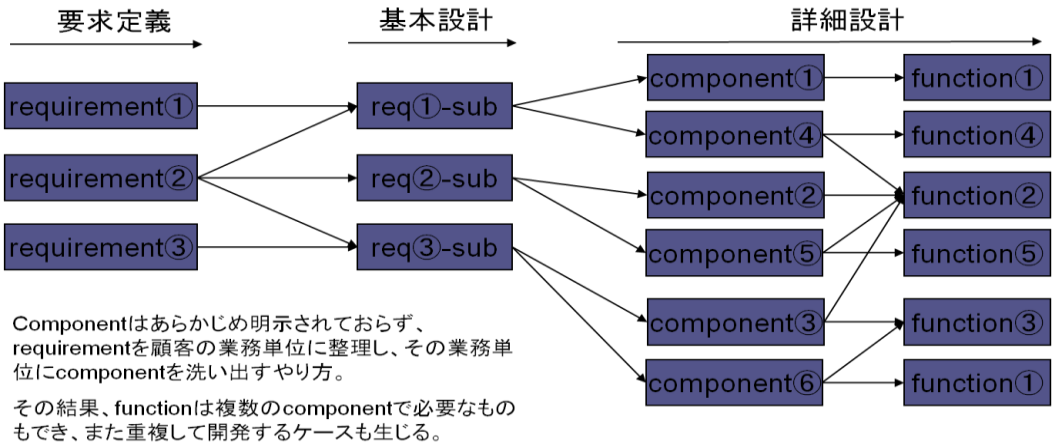


図 88　業務中心の設計

しかし、自動車開発は違うように考えられる。ハードウェアだからこそ可能なのだという考え方もできるが、図のようにアーキテクチャを構成している要素 (component) に対して要求条件をマッピングしていくような、いわばアーキテクチャ中心の設計であると解釈できる。要求を実現するにはどの component に影響があるのか、そして各 component にどのような機能を分担させればよいのか、そのようなことを基本設計の段階で行っていると考えられる。その結果として、requirement と component の関係は複雑になっても、component は洗練されたアーキテクチャであれば、function との関係性は非常にシンプルなものになる。

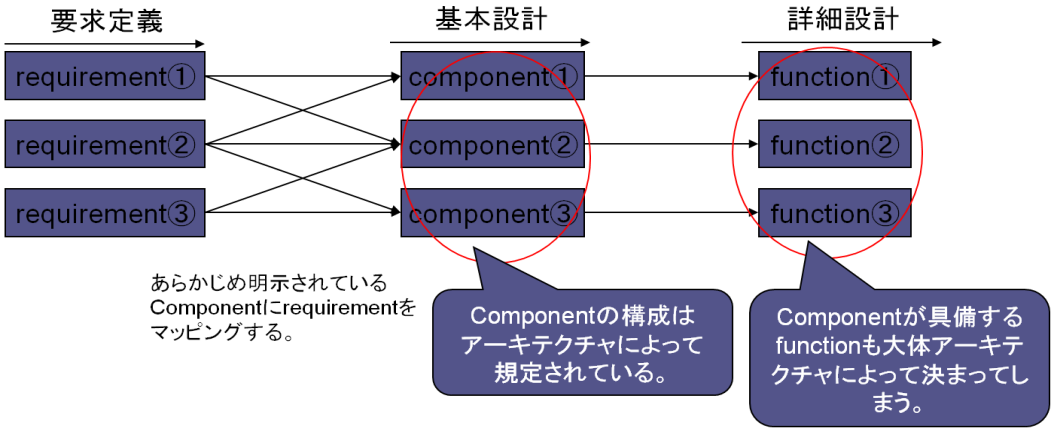


図 89　アーキテクチャ中心の設計

以上のソリューションとして 1 例を挙げるならば、情報システム開発の組織体制としては以下のような組織構成を推奨する。これにより要素技術の専門である市販のパッケージ製品メーカとの協業もやり易くなるはずである。各要素の専門を組織として設け、その担当を窓口に各市販製品メーカと情報伝達を行うのである。類似した技術も市販製品メーカで多いことから、各社製品の長所短所を把握するには情報システム開発組織もある適切な要素技術単位に専門部門を設けておくほうが適切である。この機能別組織を基本とするが、ある情報

システム開発プロジェクトが発足すると、その機能別組織から各技術要素の専門家がアサインされ、IT アーキテクトによりコーディネートされ開発業務に特化していくようなイメージである。またトヨタ社と異なる点として横軸に業種別組織を設けた。そこで顧客の業務要件を理解し、各機能別組織の技術者に伝えることを狙った（トヨタ社でいう“デザイン”に該当すると考えても問題ないが、横軸に切り出し明確化したほうが分かりやすいかと考えた)。但し、あくまでも全体コーディネーターは先ほど定義した役割を持った IT アーキテクトである。そして全体の進捗・予算管理はプロジェクトマネージャの役割である。

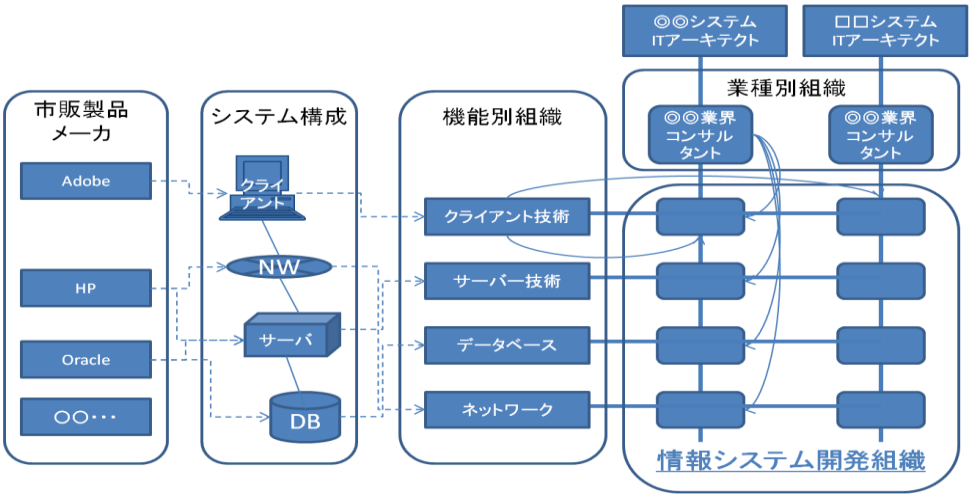


図 90　情報システム開発プロジェクトの組織構成案

現在、開発当初の多くの場合は顧客視点で体制を組むことが多い。そのため、アーキテクチャの視点からは非常に複雑に入り組んだ情報を整理しなければならない。それを今後開発当初からアーキテクチャの構成要素ごとに開発当初から組織を分けて統制する体制にすることで、マトリックスで表現するならば、各構成要素でクラスター化し、極力対角行列化することで情報伝達ルートをシンプルにする。それにより、顧客のニーズをレスポンス良く取り込み、また情報伝達ミスによる品質劣化を防止することになる。

顧客のニーズから基本要件を導き出すところまでは顧客視点で体制を組み企画提案段階（開発着手前）で実施しすることで問題ないが、それを製品への要求項目（要求仕様）に落とし込むときにはアーキテクチャ構成要素ごとに各開発担当者が目利きしていくことが必要である。

前述のとおり、新しくシステムを企画し開発することが停滞気味である現在では、既存のシステムに対する変更要求が多い。そのような場合はトヨタの事例のように Function から team の関係は対角行列化し、品質とコストのコントロールを容易にすべきである。

また needs から function も、新規の開発ではないならば比較的シンプルな関係性、つまり対角行列または三角行列で表現できるはずである。

よって、上記のような組織構成で、要求仕様をコントロールすることも含め、十分に対応が可能であろう。そういった意味では、現在が情報システム開発プロジェクトに高品質で高生産性を生み出す仕組みに再生するにはベストなタイミングであるのではないか。

おそらく、単純な機能追加のときには技術者は無意識に各要素の関係性を対角化するように requirement

として整理しているケースも多いであろう。そのような開発をプロトタイピングとして、検証し、そして改善しながら新規の開発案件に対して水平展開していくこと変革に向けたステップとしては好ましい。

11. まとめ

情報システム開発プロジェクトでは現在も将来も多様な要素が絡み、今後もその多様性が高まる傾向が強い。その時に問題になることは、情報の伝達である。設計情報など正確に且つその情報を必要とする要素に伝達することが重要であるが、多様故に複雑化し、その情報が欠落・湾曲する確率が高まる。

そこで、情報システム開発プロジェクトをひとつのシステムとして捉え、そのアーキテクチャを構成する要素を８つ抽出した。そのアーキテクチャを分析することで情報伝達を円滑にする方策を検討した。

まずは、その構成要素の関係性をネットワーク図、行列で整理するモデルを提案する。理想形は全ての行列を掛け合わせて結果が単位行列（対角行列）化されていることが望ましい。そのとき、各機能等が互いに独立し、互いに干渉することなくロバストなシステム（プロジェクト）が構築される。対角化が困難な場合でも、全行列の関係性をある要素間の限定したものに閉じる（例えば、function と artifact の関係のみに限定することや、クラスター化すること）ように改善すべきである。極限られた関係性にのみ全行列にすることで、マネジメントの注力点を絞ることが情報システム開発プロジェクト全体をコントロールするためのキーである。

また提案したモデルを用いることで生産性や品質を定量的に測ることが可能であり、今までは経験則でしか論じることが出来なかったことを理論的に分析検証することを可能とした。

このようなことを踏まえると、情報システム開発プロジェクトの立ち上げ、運営には他業界のやり方に参考になる点も多い。例えば自動車業界のチーフエンジニア制度なども勉強になる点がある。今後はそのようなアイディアを取り入れながら情報システム開発プロジェクトを一層ロバストなものとしていく必要がある。

12.　今後に向けて

＜情報システム開発プロジェクトから他業界・他業種への展開＞

本論文で紹介したモデルは情報システム開発プロジェクトに留まらず、国の政策やその他企業活動の分析・見える化などにも応用可能である。例えばこのように政府を team として activity を政策立案、予算立案、などと設定し、その成果物（artifact）を法律とする。また component は機能を実行する実体を示すものであるから、各省庁が当てはまる。そしてその各省庁で実施しているサービス・機能（function）を挙げて、それぞれを関係付けていけばよい。最終的には国民・国家の要求(requirement)のどれと結びついているのか、そこまで関係性を整理できればモデルとしては完成する。

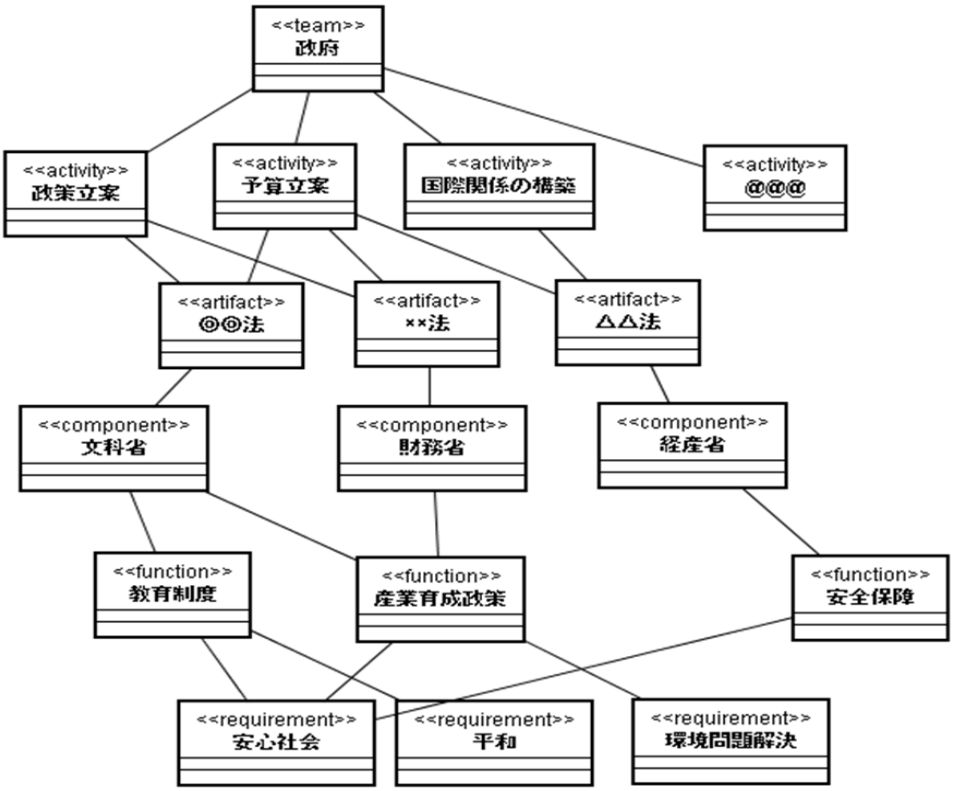


図 91　ネットワーク図（例）

政府の活動モデル

そのようにモデルの適用範囲を拡張することが可能と考える。そして今後の取り組みとして次の課題を設定したい。

顧客視点から開発側の視点（製品・サービス視点）へ変化するために、どこかで全行列は入ってしまうのではないか。おそらく、基本的に function－team 間の関係はコスト要素であるため、対角化した関係を目指してコスト低減に向けた努力をするのが企業であり、その結果、コスト要素ではない needs－requirement の間に全行列を集約することで対処している（業務改善中の企業は全行列が残っているはず）。しかし、コスト主義ではない企業では function－team の間においても全行列を組込み、それをコアコンピタンスとし、その全行列を如何にコントロールし、質を維持し向上させるか、という観点で成功している企業が、ブランドイメー

ジを確立し、企業経営にも影響を与えているように考えた。また、時代の変化にも機敏に追従し、いつの時代でも特別なブランドであるためには基本的な関係は対角行列化しており、変化に強いロバストな仕組みを形成しているのではないだろうか。そして全行列化している箇所の関係性をコントロールする何かしらの秘訣を持っているということはないだろうか。コストが多少かかったとしても、その結果、何か特別な付加価値を付けることが出来るならば企業としてはそれが特徴となり、顧客への訴求力が強まるとも考えられる。よって、どこの全行列を置くかということは企業のコアコンピタンスを作り出し、実行していく上で非常に重要な要素であると考える。

このようなことから、ひとつの仮説を考えた。顧客指向と製品・サービスを生み出すシステムのアーキテクチャ指向の間には全行列で変換するポイントが必ず存在する。その変換ポイントをどこに置くかということで企業の特徴・強みが決定付けられるのではないか。

今後はこのような仮説を元に高い品質と高い生産性を生み出す仕組み設計、つまりシステムデザインに取り組み、その仮説立証を次のテーマとして取り組んでいきたい。

＜情報システム開発プロジェクトへの導入に向けた取り組み・今後の展望＞

今回の研究では言及をしていないが、今後の情報システム開発プロジェクトに導入し展開していくにあたって下記のような取り組みを行っていく必要がある。本研究の発展に向け、下記の内容についても取り組みを今後行うこととする。

◆各開発工程におけるネットワーク図・行列の精度と詳細化手法について

ネットワーク図や、それに伴う行列は、各構成要素をどの程度の粒度で抽出するかによって異なってくる。当然開発工程の初めと終わりでは描ける図の詳細度は異なる。

今後展開していく上では、どの工程ではどの程度の粒度が適切なのか、またどの程度の粒度だと不適切であり、そのために想定すべきリスクは何なのか、ということを明らかにしておく必要がある。

◆ＱＣＤ最適解の算出方法について

各ベクトル Q,C,D それぞれの最適解を算出する方法案については本文中にも述べたとおりである。ｎ次元の空間で各ベクトル(cost, quality, delivery)を軸としたその関係性を考え、そこに出来上がる 4 面体(q-c-d-o)の点 o と平面 q-c-d の距離が最小になっているときが、各値（q、c、d）の最小値、つまり最適解ではないかと考えた。今後はこの方法案の実証検証も必要である。

◆ネットワーク図と行列の作成・マネジメントについて

上記のトピックにも関連するが、プロジェクトの企画段階では非常に粗く、数名のグループでネットワーク図を作成することになるであろう。もしそのプロジェクトが幾つかのサブシステムに分けて開発する場合は、そのサブシステム単位でもネットワーク図を作成することが有効であると考える。そのようにサブシステム分割する場合にどのようにトレーサビリティを管理してネットワーク図、行列をマネジメントしていくのか、といった観点の検討を進める必要がある。

◆時間的概念の取り込みについて

組織が時間により変化する場合のシステム品質についても考慮が必要になってくるであろう。

$$\vec{r} = A(\text{時間}) \bullet \vec{t}$$

開発プロジェクトのような社会システムでは、時間軸でシステム全体の動きが変化する場合、その行列 A の成分が時間に依存した関数として取り扱うことで、その分析が可能となる。

顧客要求 r も時々刻々と変化するが、それを処理するシステム内部は画一的になりがちであり、それが上手く時間変化できるようなシステムを作ることは重要なポイントである。

時間変化に応じたコスト、品質の変化も本モデルにより議論することが可能になれば更に有効性が高まる。

このような分析手法についても考察し、深めていきたいと考える。

13. 謝辞

本研究論文作成にあたり、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 Socio-Critical system 研究室（SCS 研） 手嶋龍一教授、保井俊之教授をはじめとする SCS 研メンバにはアイディアを詰めていくにあたって様々なアドバイス・ヒントを頂いた。また、苦手としている英文作成にあたってはスティーブ山口さん、SCS 研メンバの醍醐奈生子さんには大変お世話になりました。そして同研究科中野冠教授、西村秀和教授、佐々木正一教授、前野隆司教授、小木哲郎教授にも非常に有益なアドバイスを頂きながら完成させることができました。また勤務先である NTT コムウェア株式会社 基盤技術本部 技術 SE 部の上司・同僚共に、私の仕事と学業の両立に対し理解を示して頂き、様々な面でご協力して頂きました。

以上、この研究論文の作成にあたりご協力して下さった皆様に対し、ここに改めて深く感謝申し上げます。ありがとうございました。

引用文献

1. **北岡元.** 仕事に役立つインテリジェンス.　PHP 新書, 2008.
2. –. ビジネス・インテリジェンス.　東洋経済新報社, 2009.
3. **服部秀雄.** トヨタ自動車（株）における研究開発関連組織間の連携システム.　オペレーションズ・リサーチ, 1991.
4. **独立行政法人　情報処理推進機構　ソフトウェアエンジニアリングセンタ　所長　鶴保征城.** 情報システムの巨大化・複雑化とソフトウェア工学の役割. 2006.9.
5. **藤本教授.** 現場発の産業競争力論　－組織能力の進化とアーキテクチャの比較優位.　東大, 2007.
6. –. ものづくり論とソフトウェア　－組織能力とアーキテクチャの視点から－.　東大, 2006.9.
7. –. MMRC-J-24　アーキテクチャの比較優位に関する一考察.　東大, 2005.3.
8. –. MMRC-J-207　アーキテクチャとコーディネーションの経済分析に関する試論.　東大, 2008.3.
9. **渡辺弘美.** 米国の IT 人材市場について.　JATRO/IPA NY, 2005.3.
10. **田中克己.** IT 産業再生の針路.　日経 BP, 2008.12.
11. **中小企業基盤整備機構.** 平成 19 年度ナレッジリサーチ事業「自動車産業の多層的サプライヤー・システムと中小サプライヤーの役割」.　経営支援情報センター, 2008.3.
12. **大久保隆・持原眞理子　.** 米国政府調達における契約の種類とその実際－コントラクト・マネジメントの考慮点－　.　プロジェクトマネジメント学会誌 Vol.4　No2.　, 2002.
13. **増田直樹・今野紀雄.** 「複雑ネットワーク」とは何か.　講談社　ブルーバックス, 2006.
14. **総務省統計局.** 日本標準産業分類.（オンライン）H14 年 3 月改訂年. http://www.stat.go.jp/index/seido/sangyo/index.htm.
15. **石井浩介　飯野賢次　共著.** 価値づくり設計.　養賢堂, 2008.
16. **西口敏宏.** ネットワーク思考のすすめ.　東洋経済新報社, 2009.
17. **森健／丹治和夫.** Navigator　Vol.8「自動車部品メーカーの未来」.　ローランド・ベルガー・アンド・パートナー・ジャパン, 2003.1.
18. **社会経済生産性本部.** 米国企業における報酬制度　.　米国事務所通信, 2004.12.
19. **佐伯靖雄.** 製品開発組織と開発プロセス.　立命館経営学　第 46 号　第 5 号, 2008.1.
20. **佐藤靖.** NASA を築いた人と技術－巨大システム開発の技術文化. 東京大学出版, 2007.5.
21. **今田治.** 自動車企業のグローバル化と生産技術部門　.　立命館経営学　第 41 号　第 6 号, 2003.3.
22. **戸室健作.** 自動車産業における請負労働と分業構造.　法政大学 大原社会問題研究所雑誌 No.585, 2007.8.
23. **経済産業省.** 特定サービス産業動態統計調査（長期データ）.（オンライン）2008 年. http://www.meti.go.jp/statistics/tyo/tokusabido/result/result_1.html.

24. プロジェクト・チームの対話不足を防ぐ法. **マニュエルE．ソーサ, スティーブンD．アペンジャー , クレイグM．ロールズ.** 8 月, ダイヤモンド社　ハーバードビジネスレビュー, 2008 年.
25. **マイケル A.クマノス.** ソフトウェア企業の競争戦略.　中央経済社, 2004.12.
26. **キース・ヴァン・デル・ハイデン.** シナリオ・プランニング.　ダイヤモンド社, 1998.
27. **オリ・ブラフマン／ロッド・A・ベックストローム.** ヒトデはクモよりなぜ強い.　日経ＢＰ社, 2007.
28. **NamSuhPyo ,（中尾政之・飯野謙次・畑村洋太郎　共訳）.** 公理的設計（Axiomatic Design）.　森北出版, 2004.
29. **Manuel .E.Sosa.** Identifying Modular and　Integrative Systems and Their　Impact on Design Team　Interactions.　INSEAD, 2003.
30. −. Faculty & Research Working Paper Component Connectivity, Team Network Structure and the Attention to Technical Interfaces in Complex Product Development.：INSEAD, 2007.
31. −. DETC2005/DTM-85422　A NETWORK APPROACH TO DEFINE MODULARITY OF PRODUCT COMPONENTS.　INSEAD, 2005.
32. −. Component Modularity, Team Network Structure, and the Attendance to Technical Interdependences in Complex Product Development.　INSEAD, 2006.
33. −. A Network Approach to Define Modularity of Components in Complex Products.：INSEAD, 2007.
34. **M.CLARKROBERT.** INTELLIGENCE ANALYSIS.　CQ PRESS, 2007.
35. **LindemanUdo, MaurerMaik , BraunThomas.** Structural Complexity Management An Approach for the Field of Product Design.：Springer, 2009.
36. **Dean LeffingwellwidrigDon.** Managing Software Requirements A Unifided Approach.　：Addoison-Wesley, 2000.
37. 日本オラクル. (オンライン) http://www.oracle.com/lang/jp/index.html.
38. **産業構造審議会　情報産業分科会.** 情報サービス・ソフトウェア産業維新～魅力ある情報サービス・ソフトウェア産業の実現に向けて（案）. 2006.6.
39. **角和昌浩（IEEJ）.** シナリオプランニングの実践と理論. 2005.
40. SciLab 擬似逆行列計算プログラム.（オンライン）http://staff.aist.go.jp/toru-nakata/Gauss/Gauss2.html.
41. SciLab（数値計算ソフト）. (オンライン) http://www.scilab.org/.
42. Pajek（ネットワーク描画ツール）.（オンライン）http://vlado.fmf.uni-lj.si/pub/networks/pajek/.
43. ITpro. (オンライン) http://itpro.nikkeibp.co.jp/index.html.
44. Design Structure Matrix (DSM). DSMweb.org. (オンライン) http://www.dsmweb.org/.
45. **榮谷昭宏.** 社内標準フレームワークの活用によるＱＣＤ改善.　第 24 回　ソフトウェア品

質シンポジウム, 2005.
46. **JISA 白書分会.** 情報サービス産業白書　報告会資料. 2009.

参考資料

SciLab　計算プログラムとその実行例（１）

＜擬似逆行列の算出＞

```
-->A = [0.25 0.25 0.25 0.25;
-->0.25 0.25 0.25 0.25;
-->0.25 0.25 0.25 0.25;
-->0.25 0.25 0.25 0.25]
 A  =
            column 1 to 3
     0.25      0.25      0.25
    0.25      0.25      0.25
    0.25      0.25      0.25
    0.25      0.25      0.25

          column 4
     0.25
    0.25
    0.25
    0.25
-->
-->[U, S, V ] = svd(A)
V  =
            column 1 to 2
   - 0.5      0.8660254
  - 0.5   - 0.2886751
  - 0.5   - 0.2886751
  - 0.5   - 0.2886751

          column 3 to 4
     0.                0.
  - 0.5773503   - 0.5773503
    0.7886751   - 0.2113249
  - 0.2113249      0.7886751
S  =
          column 1 to 3
    1.      0.                0.
   0.      8.327D-17      0.
   0.      0.                0.
   0.      0.                0.

          column 4
    0.
   0.
   0.
   0.
U  =
            column 1 to 2
   - 0.5      0.8660254
  - 0.5   - 0.2886751
  - 0.5   - 0.2886751
  - 0.5   - 0.2886751
            column 3 to 4
   - 4.163D-17      0.
  - 0.5773503   - 0.5773503
    0.7886751   - 0.2113249
  - 0.2113249      0.7886751
-->
-->PinvA = pinv(A)
PinvA  =
            column 1 to 3
     0.25      0.25      0.25
    0.25      0.25      0.25
    0.25      0.25      0.25
    0.25      0.25      0.25

          column 4
    0.25
   0.25
   0.25
   0.25
```

```
-->APA = A * PinvA * A
 APA  =

        column 1 to 3
    0.25    0.25    0.25
   0.25    0.25    0.25
   0.25    0.25    0.25
   0.25    0.25    0.25

        column 4
    0.25
   0.25
   0.25
   0.25
```

```
-->PAP = PinvA * A * PinvA
 PAP  =

        column 1 to 3
    0.25    0.25    0.25
   0.25    0.25    0.25
   0.25    0.25    0.25
   0.25    0.25    0.25

        column 4
    0.25
   0.25
   0.25
   0.25
```

SciLab　計算プログラムとその実行例（２）

＜品質の算出例＞

行列 A を定義

```
-->A=[
0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001;
0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001;
0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001;
0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001,0.000000001;
0.000000001,0.000000001,0.000000001,0.000000001,0,0,0,0;
0.000000001,0.000000001,0.000000001,0.000000001,2,1,0,0;
0.000000001,0.000000001,0.000000001,0.000000001,3,2,1,1;
0.000000001,0.000000001,0.000000001,0.000000001,0,0,4,6]
```

行列 A のノルムを算出

```
-->norm(A)
 ans  =

    7.3898331
```

行列Aの固有値・固有ベクトルを計算

```
-->[D,X]=bdiag(A)
```

固有ベクトル

```
X  =

         column 1 to 2

  - 1.655D-10    9.155D-14
  - 1.655D-10    9.155D-14
  - 1.655D-10    9.155D-14
  - 1.655D-10    9.155D-14
  - 9.878D-20    3.662D-22
  - 1.626D-19    0.0000610
  - 0.1655035    0.0001526
  - 0.9436283  - 0.0001221

         column 3 to 4

  - 4.718D-10    3.725D-09
  - 4.718D-10    3.725D-09
  - 4.718D-10    3.725D-09
  - 4.718D-10    3.725D-09
  - 6.324D-18    3.725D-09
    2.072D-17  - 7.451D-09
  - 0.4718142    1.118D-08
    0.3310070  - 7.451D-09

         column 5 to 6

    0.0000161    0.0000088
    0.0000102  - 0.0000438
  - 0.0000345    0.0000300
    0.0000466    0.0000288
  - 1.430D-13  - 8.902D-14
    1.708D-13    1.064D-13
    5.499D-13    3.424D-13
  - 3.857D-13  - 2.402D-13

         column 7 to 8

  - 0.0000451  - 0.0000368
    0.0000230  - 0.0000343
    0.0000215  - 0.0000342
    0.0000265  - 0.0000051
  - 9.631D-14    4.113D-13
    1.151D-13  - 4.914D-13
```

3.703D-13　- 1.581D-12
- 2.598D-13　1.109D-12

固有値
D　=

column 1 to 3

6.7015621　0.　0.
0.　1.　0.
0.　0.　0.2984379
0.　0.　0.
0.　0.　0.
0.　0.　0.
0.　0.　0.
0.　0.　0.

column 4 to 5

0.　0.
0.　0.
0.　0.
4.000D-09　0.0000103
0.　3.604D-17
0.　0.
0.　0.
0.　0.

column 6 to 7

0.　0.
0.　0.
0.　0.
0.0000064　0.0000069
2.250D-17　2.231D-17
4.159D-20　- 7.061D-19
0.　- 2.155D-18
0.　0.

column 8

0.
0.
0.
- 0.0000296
- 4.297D-17
2.382D-17
3.934D-17
- 1.566D-17

::

行列 A の固有ベクトルを整理（a－h）

a=[- 1.655D-10 ,- 1.655D-10 ,- 1.655D-10 ,- 1.655D-10 , - 9.878D-20, - 1.626D-19 ,- 0.1655035 ,- 0.9436283]

b=[9.155D-14,9.155D-14,9.155D-14,9.155D-14,3.662D-22,0.0000610 ,0.0001526,- 0.0001221]

c=[- 4.718D-10,- 4.718D-10,- 4.718D-10,- 4.718D-10,- 6.324D-18 ,2.072D-17 ,- 0.4718142, 0.3310070]

d=[3.725D-09,3.725D-09,3.725D-09,3.725D-09,3.725D-09,- 7.451D-09 ,1.118D-08 ,- 7.451D-09]

e=[0.0000161,0.0000102,- 0.0000345,0.0000466,- 1.430D-13, 1.708D-13 , 5.499D-13, - 3.857D-13]

f=[0.0000088,- 0.0000438,0.0000300, 0.0000288,- 8.902D-14, 1.064D-13,3.424D-13 ,- 2.402D-13]

g=[- 0.0000451,0.0000230,0.0000215,0.0000265 ,- 9.631D-14,1.151D-13,3.703D-13 ,- 2.598D-13]

h=[- 0.0000368 ,- 0.0000343,- 0.0000342,- 0.0000051, 4.113D-13 ,- 4.914D-13,- 1.581D-12 ,1.109D-12]

固有値・固有ベクトルで作られるベクトル空間を求める。

i=a*6.7015621 +b*1 +c*0.2984379 +d*4.000D-09 +e*3.604D-17 +f*4.159D-20 +g* - 2.155D-18 +h*- 1.566D-17

よって、

i=[1.250D-09,1.250D-09,1.250D-09,1.250D-09,1.23

5D-17,0.000061,- 1.2497866,- 6.2251207]

となる。また、同じ次数の対角行列で作られるベク

トルは

j=[1,1,1,1,1,1,1,1]

であるから、i と j の内積は

i*j'=- 7.4748463

またベクトル i の大きさは

i*i'=40.314094

であり、平方根

Sqrt(i*i')= 6.3493381

と算出される。

ベクトル h の大きさは 8 の平方根を同様に求め

2.8284271

よ っ て 、 cos θ ＝ - 7.4748463/(6.3493381

*2.8284271)= - 0.4162256

品 質 ＝ - 0.4162256 × （ 1/7.3898331 ） ＝ -

0.0563241