情報処理学会論文誌 Vol.55 No.5 1453–1470 (May 2014)

# 設計情報の写像経路に着目したソフトウェア開発プロジェクトの複雑性指標の提案

榮谷 昭宏[1,2,a)] 牧野 泰才[3] 前野 隆司[3]

受付日 2013年8月19日, 採録日 2014年2月14日

**概要**：本研究ではソフトウェア開発プロジェクトの要素間の複雑な関係をシンプルに表すモデルを提示し，その有用性を示すことを目的とする．本モデルでは，設計情報の写像経路に着目したプロジェクトの複雑性指標を用いる．そして，その複雑性指標と生産性（1 人あたりの平均作業日数）および開発工数が相関することを実証的に検証する．これにより，プロジェクトの生産性向上や工数削減を実現するためには，設計情報を写像する経路の冗長性を低減させることが必要であり，そのためには提示したモデルが有用であることを示す．そのモデルとは，公理的設計論に基づいて構築され，ソフトウェア開発プロジェクトにおけるプロセスやプロダクトを構成する要素間の関連の複雑さを表している．その複雑性は，設計情報の写像経路数や各経路の写像の難しさから，プロジェクト全体の写像の難しさを数値化したものである．そして提示したモデルの有効性を示すために，実際に 21 のソフトウェア開発プロジェクトからデータを収集することで検証を行った．その結果，複雑性指標と生産性や開発工数との間に強い相関があることが分かった．これらの検証結果から，本研究で示したモデルの複雑性指標を低減させることはプロジェクトの生産性向上や工数削減に有効であることを確認した．

**キーワード**：複雑性，生産性，写像経路，ソフトウェア開発プロジェクト，開発工数

# Software Development Project Complexity Index Approach Focused on Mapping Pathways of Design Information

Akihiro Sakaedani[1,2,a)] Yasutoshi Makino[3] Takashi Maeno[3]

Received: August 19, 2013, Accepted: February 14, 2014

***Abstract:*** Software development project members communicate with each other by various means and various subjects. The management for such complexities of the communication makes an effect on project quality and development cost. As pointed out by researchers, the communication in a project makes a real influence on quality and cost. But the researchers have not assessed the project complexity yet. In particular, the correlation with productivity and development cost remains unproved. In this paper, we propose the idea of the complexity index focused on mapping pathways of design information in software development project. This research establishes a correlation between the complexity and productivity (average work days per person), development cost (man-hour). We develop the project model using axiomatic design theory. A large portion of the software development activity is made up of transforming design information. Simpler of the mapping pathways is better for productivity and development cost. This research proposed the complexity index for mapping pathways. And we proved the validity of the complexity index by discussion of a case of software development project.

***Keywords:*** complexity, productivity, mapping pathways, software development project, development cost

## 1. はじめに

### 1.1 ソフトウェア開発プロジェクトにおける複雑性のマネジメントの現状

ソフトウェア開発プロジェクトは，設計情報を開発者間で伝達し，詳細化し，ソースコードを書くという，設計情報をやりとりするコストが開発費用の大半を占める．そのため，開発要員を増やしても，必ずしもプロジェクト全体の生産性が向上するわけではない．逆に生産性が悪くなるケースも多い．なぜならば，開発要員の増加は開発要員間のコミュニケーションを複雑化させるためである [1]．このため，複雑なコミュニケーションが必要な作業内容を改善しなければ，効率的な開発（生産性向上や工数削減）は実現できない．

開発プロジェクトにおけるコミュニケーションに関する先行研究は，以下の 2 つの観点から分類することができる．1 つ目はコミュニケーションによる情報伝達の難しさの要因に関する研究である．2 つ目は適切な相手とのコミュニケーションの有無に関する研究である．まず，1 つ目の観点として，開発プロジェクトのコミュニケーションを改善するために，犬塚 [2], [3] はソフトウェア開発プロジェクトの開発者間の情報のやりとりを調査した．その結果，プロジェクト内でやりとりされる情報の質や量，媒体，送受信者の能力差によって情報伝達の難易度が変わることを示した．一方，2 つ目の観点として，Sosa ら [4] は，エンジン開発プロジェクトにおいて，部品間に相互依存があるならば，インタフェースを決めるためにその開発者間のコミュニケーションも必要となることに着眼した研究を行った．すなわち，開発者同士のコミュニケーションと同時に，エンジンを構成する部品間の相互依存を調査した．そして，部品間の相互依存と，各部品の開発者間で行われるコミュニケーションの有無を比較する方法を提案している．以上より，無駄なコミュニケーションや必須のコミュニケーションの欠如等プロジェクト内コミュニケーション状態を把握することが，品質や生産性の向上に有益であると述べている．これらの研究は，プロジェクト内のコミュニケーションについて以下の点を定性的に明らかにしている．すなわち，プロジェクト内の複雑なコミュニケーションでは，そもそものコミュニケーションの有無，およびそのときやりとりされる情報の質や量，伝える媒体，送受信者の能力差が，開発されるプロダクトの品質やプロジェクトの生産性・工数に影響を与えるということである．

しかし，このどちらの先行研究も，なぜ，開発作業では複雑なコミュニケーションが必要となるのか明らかにされていない．これは，他の先行研究でも同様である．ソフトウェア開発プロジェクトのプロセスに特化したものではないが，前述の犬塚と同様，von Hippel [5] や Draft ら [6] も人と人とのコミュニケーションについて研究を行っている．そして，その情報の質や量，媒体，送受信者の能力差に関連した情報伝達の難しさを指摘している．しかし，これらの研究では，コミュニケーションが複雑化する仕組みは明らかにされていない．一方，前述の Sosa らの研究と同様に，開発プロセスとプロダクトの関連性については Eppinger ら [7], [8], [9], [10] や Danilovic ら [11], [12] も取り組んでおり，マトリックス（Design Structure Mtrix [13]）を用いた方法論を提案している．しかし，これらのプロセスとプロダクトの両面からの研究においても，コミュニケーションが複雑化する仕組みは明らかにされていない．実際，ソフトウェア開発では，良い開発プロセス（作り方）が良いプロダクト（製品）を生むことを前提に考えられているものの，プロダクトとプロセスの関連は十分に解明されていない [14]．すなわち，プロセスにおけるコミュニケーションが，プロダクトにどのように関連しているのか，そして，その関連性がコミュニケーションの複雑化にどのように寄与しているのか，明らかにされていない．さらに，その関連性が，プロジェクトの生産性・工数にどのように影響を及ぼすのか，その仕組みも明らかにされていない．

また，上記の研究では，コミュニケーションの複雑さとプロジェクトの生産性や工数等の各種マネジメント指標との関係については明確に示されていない．実際の開発プロジェクトにおいても White ら [15] が示したようにプロジェクトで利用される主要なマネジメントツールはガントチャートによる作業の進捗管理ツールであり，複雑性のマネジメントに関するツールは活用されていない．

このため，筆者らは過去にプロジェクトの複雑性に関する研究を行ってきた．まず，先行研究では複雑性を相互依存性と難易度の積と定義し，プロジェクトを構成する要素間の複雑な関係を指標化した．その指標値の大小により，プロジェクトの複雑さの度合いを知ることはできた．まず，複雑性の概念の初期案を提示し [16]，アニメ・CG 製作プロジェクトの複雑性を机上で検証した [17]．次に算出方法に改良を加え，プロジェクトでの活用方法を机上のモデルを用いて説明した [18]．また複数プロジェクト間で部品を流用する場合の効果について，その分析方法を提案した [19]．最後に実プロジェクトにおける設計課題数と複雑性の相関を検証した [20]．しかし，筆者らの過去の研究における複雑性の定義ではプロジェクトにおける設計情報の写像経路

[1] 慶應義塾大学大学院システムデザイン・マネジメント研究科博士課程
Doctoral Program, Graduate School of System Design and Management, Keio University, Yokohama, Kanagawa 223–8526, Japan
[2] NTT コムウェア株式会社
NTT COMWARE CORPORATION, Minato, Tokyo 108–8019, Japan
[3] 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
Graduate School of System Design and Management, Keio University, Yokohama, Kanagawa 223–8526, Japan
[a)] a.sakaedani@sdm.keio.ac.jp

数や各経路における写像の難しさを明らかにしていない．したがって，設計情報のやりとりがコミュニケーションを複雑化させるか否かの説明には至っていない．当然，写像経路と生産性や開発工数の関連は考慮していなかった．また，その複雑性と生産性の相関については言及しているものの，開発工数との相関については言及していない．しかも，生産性と複雑性の相関は机上モデルによる説明にとどまり，実プロジェクトでの検証は行っていない．

なお，複雑性について Lindeman ら [21] では工学の分野において標準的な定義は存在しないと述べている．同書では，複雑性は要素間の相互作用から生じるという主張もある [22]．一方，同書では Ehrlenspiel [23] は技術システムにおいてはその構成要素の量とそれぞれの結合性に依存するが，モノの複雑性は変数の量，結合度，凝縮性，運動量に依存するという主張も紹介している．このように複雑性の定義は一意に定まったものではなく，個々でその定義を行っている．このため，従来は複雑性の定義が明確ではなかった．

以上のように，従来の研究では，コミュニケーションの難しさを生じる要因，コミュニケーションの有無が及ぼす影響について研究が行われ，コミュニケーションがプロジェクトの品質や生産性に影響を及ぼすことは定性的に指摘されていた．しかし，ソフトウェア開発プロジェクトのプロセスやプロダクトを構成する要素間の複雑な関係を明確かつシンプルに表した，実プロジェクトにおいて有効な複雑性の研究は行われていなかった．

### 1.2 本研究の目的と概要

本研究では，ソフトウェア開発プロジェクトの要素間の複雑な関係をシンプルに表すモデルを提示し，その有用性を示すことを目的とする．本モデルでは，設計情報の写像経路に着目したプロジェクトの複雑性指標を用いる．そして，その複雑性指標と生産性（1 人あたりの平均作業日数）および開発工数が相関することを，実際の開発プロジェクトを対象に実証的に検証する．これにより，プロジェクトの生産性向上や工数削減を実現するためには設計情報を写像する経路の冗長性を低減させることが必要であり，そのためには提示したモデルが有用であることを示す．

以下に，本研究の概要を述べる．まず，公理的設計論 [24] における設計情報の写像経路を簡素にモデル化することがコミュニケーションの複雑さを緩和し，コスト低減につながると考えた．そこで 2 章では，公理的設計論を発展させたプロジェクトモデルとその複雑性指標を構築する．提案した複雑性指標は生産性および開発コストと相関する指標であることはもちろん，以下の観点で先行研究の課題を解決している．まず第 1 に，プロジェクトを構成する要素間の相互依存性だけでなく，設計情報の写像の難しさも考慮した複雑性指標を導出する．そして，開発するプロセスだけでなく開発するプロダクトとの関連をその写像経路で示すことで先行研究の課題に取り組む．3 章では，複雑性指標の具体的な算出方法を紹介する．2 章で紹介したモデルを行列で表現する方法を示す．その行列計算によって，プロジェクトで形成される写像経路の本数が算出できることを示す．また，各経路の写像の難しさの算出方法も同時に示す．さらに，4 章において，実際にゲームソフトウェア開発を行った 21 の開発プロジェクトから複雑性と生産性，開発工数の相関を実証的に検証する．その結果，提案した複雑性指標と生産性および開発工数に強い相関関係があることを示す．5 章では，検証結果を考察し，開発工数や生産性（1 人あたりの平均作業日数）の大半が写像経路の冗長性に起因していることを説明する．

なお，生産性とは，本来，生産物の量を投入物の量で割った値として表される．本研究では，さらに生産性を 2 つの要素に分解し，① 1 人あたりの平均作業日数（全工数/開発要員数）と ② 1 人日あたりの成果物量（生産物の量/全工数）の積で表されると考えた．しかし，本研究における生産物の 1 つである音声データは，一般的にソフトウェア開発で用いられる生産物の量であるソースコード行数と同じ単位で測ることができず，プロジェクトのすべての生産物の量を 1 つの数値で規定することができなかった．したがって，② を規定できなかった．そのため，以降では，生産性を狭義にとらえ，① 1 人あたりの平均作業日数という作業効率を表す指標のみを生産性として定義する．

## 2. ソフトウェア開発プロジェクトのモデル化とその複雑性指標の構築

まず，本研究ではソフトウェア開発プロジェクトをどのようにとらえるのか，その枠組みとなるプロジェクトのモデルについて説明する．

### 2.1 公理的設計論の定義と本研究における拡張点

公理的設計論とは，独立公理（具備する機能は独立性を高く設計した方が良い設計である）と情報公理（設計に必要な情報量は少ない方が良い設計である）の 2 つの公理からなる，良い設計を行うための設計方法論である [24]．その理論の中で，設計とは顧客領域，機能領域，実体領域，プロセス領域の 4 つの領域間における設計情報の写像過程と述べられている．顧客領域とは顧客が期待する商品等，顧客のニーズを扱う領域である．また機能領域は顧客の要求仕様が要求機能と制約条件で示された領域である．実体領域は設計解を示す領域である．そしてプロセス領域は生産条件を示す領域である．

本理論では，機械工学の分野だけでなく，ソフトウェア自体やビジネスの構造等も利用可能な設計方法を説明している．以上のように一般論としての設計理論であるため，上記のように各領域の意味付けはされているものの，具

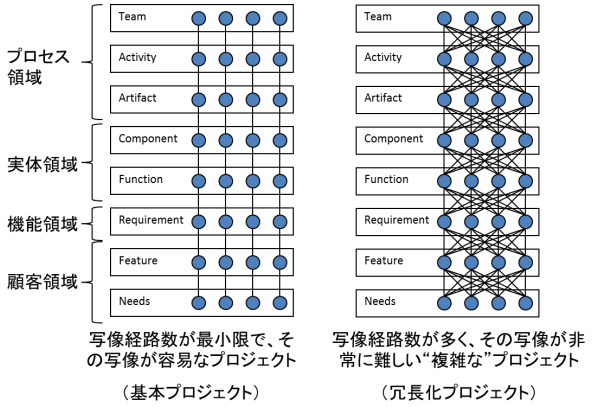



図 1 ソフトウェア開発プロジェクトの情報写像モデル
**Fig. 1** Information mapping model of software development project.

体的な要素は規定されていない．一方，本研究は，ソフトウェア開発プロジェクトを研究対象としている．そこで，次節では，公理的設計論に基づいたソフトウェア開発プロジェクトの構成要素を規定し，写像関係をモデル化する．

### 2.2 ソフトウェア開発プロジェクトのモデル化

プロセス領域に開発者（Team），作業（Activity），成果物（Artifact），実体領域に機能（Function），部品（Component），機能領域に要求仕様（Requirement），顧客領域にニーズ（Needs），基本要件（Feature）を要素として抽出し，それらの要素間で写像が行われる経路をモデル化した．

**図 1** に示した丸印はプロジェクトを構成する個々の要素である．たとえば，プロセス領域の Team においては，アーキテクトやプログラマといった開発者の役割を個々の丸印で表現している．また，図中に示した実線は個々の要素間の関連を示している．実線で結ばれた要素間で設計情報が写像されることを表している．

以下で各要素の関連について説明を行う．Team とはプロジェクト内で分担される様々な役割ごとに配置されるチームまたは個人を指す．たとえば，前述の例のとおりアーキテクトやプログラマ等が該当する．Activity は Team が実施する仕事であり，Artifact を生み出す．たとえば仕様書の作成等が該当する．Artifact は Activity により生産されるものであり，たとえば設計書等を指す．以上をまとめると Team が Activity を行うことで Artifact を生み出すという関係にある．したがって，プロセス領域内の要素の関連は，図に示したように Team と Activity を実線で結び，Activity と Artifact を実線で結んだ．Team は何かしらの Activity を行うことで Artifact を参照または更新するのであって，Activity を介さずに Artifact と関連することはないと考えた．次に，Component は Artifact を実装したものであり，Function を有する．Function は Component の持つ役割を示し，これにより Requirement を実現する．したがって，実体領域の要素の関連はプロセス領域の Artifact と Component を実線で結び，Component と Function を実線で結び，さらに次で説明する機能領域にある Requirement と Function を実線で結んだ．あくまでも Component が保有する Function が Requirement を実現するので，Component が Requirement を実現するわけではない．そのため Component と Requirement との関連付けはないと考えた．また同様に Function は Component に具備されてはじめて有効となるものなので，Component を介さずに Function と Artifact が直接関連することはないと考えた．機能領域にある Requirement は，システムとして具備すべき Function を規定したものである．たとえば “システムにより△△を行い□□を出力すること．” のように記述される．Requirement は次に説明するように Feature を具体化したものなので Feature と実線で結んでいる [25]．さらに Feature は，システムで実現すべき事項を規定しており，たとえば “システムは ×× を実現すること．” のように記述される．Feature をより具体化したものが Requirement である．Needs は，たとえば “我々は○○が欲しい．” というように顧客が欲しているものを示す．以上から，顧客領域では Feature と Needs を実線で結んだ [25]．また，その具体化の度合いから Needs は Feature を介してのみ Requirement と関連を持つ [25] と考えた．

### 2.3 プロジェクトモデル内の写像過程と生産性および開発工数

本節では，図 1 に示したモデルを用いてプロジェクト内の写像過程を説明する．前述のとおり，先行研究では，① コミュニケーション時にやりとりされる情報の質や量，伝える媒体，送受信者の能力差，および ② そもそものコミュニケーションの有無が，生産性や開発工数に影響を及ぼすことが個々に明らかにされている．これら ① ② で述べられているのは人同士の情報伝達を前提としているが，この考え方は，本研究における上記の要素間の写像においても以降で述べるように応用が可能と考えた．その結果，プロジェクトの写像過程では，個々の要素の持つ写像の難しさと写像経路の冗長性の 2 点を個別にではなく，合わせて考慮することが，生産性や開発工数に影響をとらえるうえで必要であることが明らかになった．

#### 2.3.1 要素の持つ写像の難しさと生産性および開発工数

まず，上記 ① について，本研究で用いた各要素が持つ写像の難しさを生じる特性として応用し，**表 1** にまとめた．

このようにプロジェクトの構成要素は，人同士のコミュニケーションでなくても，やりとりされる情報の質や量，能力によって，写像を難しくする特性を持つと考えた．

以降で各要素の持つ特性が，どのように写像を難しくするのか述べる．たとえば，開発者（Team）のスキルや経験は，作業（Activity）に影響する．なぜならば，スキルや経験がなければ，そもそも作業（Activity）として行うべき

表 1 プロジェクトを構成する要素と写像における特性
Table 1 Project model element and Feature for mapping.

| 領域 | 要素 | 写像の難しさを決める特性 |
|---|---|---|
| プロセス領域 | 開発者（Team） | “開発者”のスキル、経験は“作業”の効率性に影響する。スキルが低く、経験が少ない程、写像が難しい。 |
| | 作業（Activity） | “作業”の量や難しさは“成果物“の品質に影響する。量が多く、難しい程、写像が難しい。 |
| | 成果物（Artifact） | 成果物の量や品質は部品の質や量に影響する。量が多く、質が悪い程、写像が難しい。 |
| 実体領域 | 部品（Component） | 部品の質や量は保有する機能の質や量に影響する。質が悪く、量が多い程、写像が難しい。 |
| | 機能（Function） | 機能の質や量は要求仕様の充足性に影響する。質が悪く、量が多い程、写像が難しい。 |
| 機能領域 | 要求仕様（Requirement） | 要求仕様の項目数や曖昧さやは基本要件の充足性に影響する。曖昧で項目が多い程、写像が難しい。 |
| 顧客領域 | 基本要件（Feature） | 基本要件の曖昧さや要件数はニーズの充足性に影響する。曖昧で量が多い程、写像が難しい。 |
| | ニーズ（Needs） | ニーズの曖昧さやその数は、基本要件の曖昧さを生じさせる。曖昧で数が多いほど写像が難しい。 |

1つ1つの動作すら，正確に行うことができないからである．また，作業（Activity）自体の量や質によっても，開発者（Team）のスキルや経験を活かした作業（Activity）を行えるかどうかが決まってしまう．当然，正確な動作ができなければ正確に動作ができるまでやり直し（写像のやり直し）が生じるため，生産性が低下し，それにともなった開発工数の増大が起きる．同様に，作業（Activity）の中で行う動作の1つ1つの質は，当然成果物（Artifact）の質に影響を及ぼす．もし作業（Activity）量が多ければ，1つ1つの動作の質も低下することが考えられ，それも同様に成果物（Artifact）の質に影響を及ぼす．質が悪ければ，何度も質が良くなるまで写像を繰り返すことになり，これも生産性の低下を招き，それにともなって開発工数が増加する．成果物（Artifact）の量が多い場合や出来上がりに高い質を求められている場合等，作業（Activity）にも高い正確性を求められるため，同様に写像を繰り返すことが想定される．その結果，生産性や開発工数に影響することは言うまでもない．また，成果物（Artifact）と部品（Component）の関係についても同様である．成果物（Artifact）の質や量は部品（Component）に影響を及ぼし，そのため写像過程の手戻りを生む．部品（Component）と機能（Function），機能（Function）と要求仕様（Requirement），要求仕様（Requirement）と基本要件（Feature），基本要件（Feature）とニーズ（Needs）についても同様である．

### 2.3.2 写像経路の冗長性と生産性および開発工数

前項では，プロジェクトを構成する要素の特性による写像の難しさと，それにともなって生産性や開発工数に影響が生じることを述べた．しかし，前述の ② に関してはまだ述べていない．② では，開発されるプロダクトの部品構造を基準として，コミュニケーションをすべき開発者を規定していた．そして，その開発者同士が情報のやりとりをしているか否か，または本来ならばやりとりする必要のない開発者同士が情報のやりとりをしているか否か，というコミュニケーションの冗長性が生産性や開発工数に影響を及ぼすと述べている．本研究においては，写像経路は冗長な経路なのか否か，つまり必要な要素間のみを写像する経路になっているのか，という経路の冗長性の観点で ② を応用することを考えた．

以下に写像経路の冗長性の影響を述べる．たとえば，写像経路がある要素で交じりあうことがなく，つまり写像経路に冗長性がまったくなく，Team から Needs まで写像するだけならば，前項で述べたように，その写像の正確性は表 1 で示した各要素の特性だけで決まる．生産性や開発工数も同様である．図 1 の左図に示したように写像経路が途中で分岐することなく，Team のある 1 つの要素を示す丸印から Needs の 1 つの要素を示す丸印まで，1 本の写像経路で表される場合である．しかし，図 1 の右図に示したように写像経路が途中で分岐し，Team のある 1 つの要素を示す丸印から Activity の全要素の丸印と実線で結ばれ，同様に Needs の要素まで複数の関連付けがなされている場合は，個々の要素の特性だけでは写像の正確性は決まらない．なぜなら，たとえば，Artifact のある 1 つの丸印ともう 1 つ別の丸印，それらと Component の 1 つの丸印に写像関係が存在している場合（つまり，2 つの写像経路が存在している場合），Artifact から Component へ，もし前項で述べた特性により一方の経路で写像が正確に行われなければ，Component の要素は必要な設計情報を欠いてしまうこととなるからである．その結果，冗長な写像経路が増えるほど，生産性が低下し，それにともない開発工数の増大が見込まれる．以上より，本研究で示した要素間の写像経路の冗長性は生産性や開発工数に影響を及ぼすといえる．

上記をふまえ，本研究では，写像経路の冗長性と，前項で述べた個々の要素の特性も，正確な写像を行ううえで同時に考慮すべきであると考えた．したがって，以降では，これらの 2 つの観点からプロジェクトの複雑性を導出していく．

## 2.4 ソフトウェア開発プロジェクトの複雑性指標の導出

プロジェクトを構成する写像経路と，その写像経路の写像の難しさを用い，プロジェクトの生産性と相関する複雑性指標を定義する．前節で述べたように，要素間の関係が 1 対 1 ならばその写像経路の写像の難しさが生産性と相関する．また，写像経路の冗長さが増すほど生産性が低下する．よって，プロジェクトの複雑性を以下の式 (1) で定義する．そして，写像経路の複雑性とプロジェクトの開発作業の生産性（開発要員 1 人あたりの平均作業日数）の関係を式 (2) により定義する．

なお，各変数は以下のように定義する．

$\alpha$; 複雑性
$\beta$; 基本プロジェクトの写像の難しさ
$\gamma$; 冗長化したプロジェクトの写像の難しさ
$i$; 開発要員数
$k$; 1 本平均の写像の難しさ

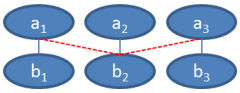



図 2 並行経路数と冗長な経路数の定義
**Fig. 2** Number of parallel and redundant pathway.

$l$; 全経路数
$m$; 並行経路数
$n$; 冗長な経路数
$C$; 全開発工数
$D$; 1 人あたりの平均作業日数
$W$; 複雑性指標の総和
以降では，上記変数を用いる．

$$\alpha = k \times m \times \left(1 + \frac{n}{m}\right) \tag{1}$$

$$\alpha \propto D \tag{2}$$

並行経路数（$m$）とは**図 2** における実線で示した経路の数である．図 1 の左図と同様，列をまたがらない場合を意味する．冗長な経路数（$n$）とは図 2 における点線で示した経路の数である．すなわち，列をまたがった写像経路数を意味する．

### 2.5 複雑性の定義式の変形

複雑性を式 (1) で定義したが，実際にプロジェクトの複雑性を求めることは難しい．なぜならば，そもそも経路 1 本平均の写像の難しさが分からないからである．しかし，式 (1) を変形すると “個々の経路の持つ写像の難しさの平均値 ($k$) × 全経路数 ($l$)” となる．言い換えれば，複雑性とは，個々の経路の写像の難しさの和となる．以下にそれを示す．

$$\begin{aligned}\alpha &= k \times m \times \left(1 + \frac{n}{m}\right)\\ &= k \times m \times \left(\frac{m+n}{m}\right)\\ &= k \times m \times \frac{l}{m}\\ &= k \times l\\ &= \sum_{1}^{\text{全経路数}} (\text{冗長化プロジェクトの各経路の写像の難しさ})\\ &= \gamma \end{aligned} \tag{3}$$

### 2.6 複雑性からの逆算方法（基本プロジェクトの求め方）

複雑性は，冗長化したプロジェクトにおける写像の難しさ（$\gamma$）に等しい．したがって，冗長化したプロジェクトの写像の難しさが分かれば，冗長化していないプロジェクトの写像の難しさも逆算できる．冗長化していないプロジェクトの写像の難しさを算出できれば，どの程度冗長化によってプロジェクトの複雑性が増したのか測定することができる．

まず，元のプロジェクトの写像の難しさを $\beta$ とすると式 (3) は次式となる．

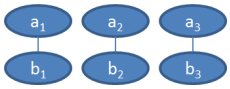



図 3 基本プロジェクトの定義
**Fig. 3** Basic project model.

$$\alpha = k \times m \times \left(1 + \frac{n}{m}\right) = \beta \times \frac{l}{m} = \gamma \tag{4}$$

式 (4) における基本プロジェクト（$\beta$）は，**図 3** のように，冗長な経路が存在しない状態と定義する．

したがって，式 (4) より，基本プロジェクトは以下のように算出できる．

$$\beta = \gamma \times \frac{m}{l} \tag{5}$$

$\alpha$（$= \gamma$）と $\beta$ を比較することで，プロジェクトの冗長化による複雑性の増分を測ることができる．

### 2.7 複雑性と開発工数の関係式

本節では開発工数と複雑性の関係式を導出する．式 (2) より，複雑性は，1 人あたりの平均作業日数と相関すると考えられている．したがって，開発工数（1 人あたりの平均作業日数と要員数の積）は，複雑性に要員数を掛け合わせたものと相関すると考えられる．そのとき，プロジェクト全体の複雑性は，プロジェクトに存在する全経路で，すべての要員が携わって写像していると考え，算出されている．そして，その複雑性と 1 人あたりの平均作業日数が相関するとしている．しかし，実際のプロジェクトでは，開発中に必ずしも全経路の写像にすべての要員が携わっているとは限らない．したがって，開発工数を算出するうえでは，開発期間中のある時点において，写像されている経路数に対して，その写像に携わっている開発要員数を考慮した方が精度を高く開発工数を求めることができると考えた．そこで，開発期間中のある時点での写像経路数に対する複雑性と，それに携わる開発要員数をもとに，開発工数を積み上げ，全開発工数を算出する方法を以下に述べる．

まず，開発期間中のある時点 $t$ で，冗長な写像経路数が $j$ 本のときの複雑性，1 人あたりの平均作業日数，開発要員数，開発工数等を
$\alpha_j$; $j$ 本のときの複雑性
$C_j$; $j$ 本時の開発工数（人日）
$D_j$; $j$ 本時の 1 人あたりの平均作業日数（日）
$i_j$; $j$ 本時の開発要員数（人）
$W$; 複雑性指標の総和
のように定義すると，式 (2) より，$t$ 時点の $j$ 本の写像経路で行う開発工数は

$$C_j = D_j \times i_j \propto \alpha_j \times i_j \tag{6}$$

と表される．上記式 (6) に対して，開発期間中，プロジェクトとしてとりうる経路数は 0 から $n$ 本までであるので，

その総和は次式となる．

$$C = \sum_{0}^{n}(D_j \times i_j) \propto \sum_{0}^{n}(\alpha_j \times i_j) \tag{7}$$

すなわち，全開発工数（$C$）と複雑性（$\alpha$）は式 (7) として表されると考えた．

また，式 (7) は，開発期間中，開発要員数が変化しない場合は，$i_j$ が一定値 $i$ となるので，以下のように表される．

$$C = \sum_{0}^{i} D_j \times i \propto \sum_{0}^{i} a_j \times i = k \times m \times i \times \left(n + \frac{n^2}{2m}\right)$$
$$= W \times i \tag{8}$$
$$C \propto W \times i \tag{8'}$$

開発期間中，本研究では開発要員数（$i$）は一定であったので，全開発期間を通した開発工数との関係を式 (8) のように設定した．これらをふまえると式 (8) は，**図 4** のように表される．図 4 は複雑性（$\alpha$）と冗長な経路数（$n$）の関係式と x 軸，y 軸で作られる台形を示している．複雑性指標の総和（$W$）とはこの台形の面積を指す．式 (8′) より，プロジェクトの全開発工数（$C$）は，この面積と要員数（$i$）の積である．そして，台形の中に描かれている縦長の長方形は，冗長な経路数 $j$ 本のときの複雑性を示す．また，式 (1) の y 切片は図中の基本プロジェクトの写像の難しさ（$\beta$）を示しているので，冗長性がないときの生産性に等しい．すなわち，冗長性がない場合の複雑性指標の総和は，y 切片を縦，経路数を横とする長方形の面積に等しい．次に，冗長性に起因する生産性（1 人あたりの平均作業日数）は冗長なプロジェクトの生産性（冗長な経路数 $n$ 本のときの y 軸の値）から基本プロジェクトの生産性（y 切片の値）を差し引いた値となる．それは図 4 の上三角形の高さで示される．すなわち，冗長性に起因する複雑性指標の総和は上三角形で示される面積に等しい．複雑性指標の総和と開発要員数の積はプロジェクトの全開発工数である．したがって，冗長性に起因する開発工数は図の上三角形の面積に要員数を掛け合わせることによって求めることができる．同様に冗長性がない場合の開発工数は長方形の面積と要員数の積で求まる．

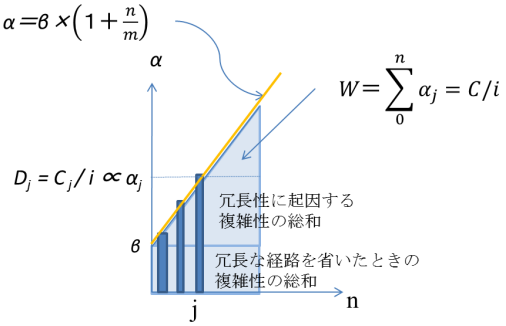



**図 4** 複雑性指標と開発工数

**Fig. 4** Project complexity and man-hour.

## 3. 写像経路本数と写像の難しさの算出方法

プロジェクトのモデルとその複雑性の算出方法は前章で述べた．しかし，その算出に用いる写像経路数や写像の難しさの計算方法は，まだ述べられていない．そこで本章では，その写像経路数や難しさの算出方法を説明する．

### 3.1 行列を用いた算出の基本原理

行列を用いることで，経路数，写像の難しさの算出が可能である．その算出方法を次に示す簡易モデルで説明する．

**図 5** の上図で示したモデルの丸印（$a_1$～$c_2$）はプロジェクトを構成する要素を示す．丸印どうしを結んだ実線は写像関係の有無を示す．実線上に付与した数値はその写像経路における写像の難しさを示している．このモデルに描かれている写像経路をすべて抽出したものが図 5 の下図である．$a_1$ から $c_1$ への写像は 1 本のみ，$a_1$ から $c_2$ への写像経路は 2 本，$a_2$ から $c_1$ への写像経路は 1 本，$a_2$ から $c_2$ への写像経路は 2 本，それぞれ具体的な写像経路とともに示している．また，各モデルの写像の難しさの算出式が各モデルの下に示してある．各写像経路の写像の難しさの積がその算出式の解となっている．また，その積の合計値と前述の写像経路数を矢印の先に，あわせて記載した．

しかし，このように，図から個々の写像経路を抽出したうえで写像経路数や写像の難しさを算出することは，モデルが複雑化するほど困難な作業となってしまう．そこで，行列を用いて算出する方法を考えた．まず写像経路数の算出方法は次式（式 (9)）となる．

$$\begin{array}{c|cc} & b_1 & b_2 \\ c_1 & 1 & 0 \\ c_2 & 1 & 1 \end{array} \times \begin{array}{c|cc} & a_1 & a_2 \\ b_1 & 1 & 1 \\ b_2 & 1 & 1 \end{array} = \begin{array}{c|cc} & a_1 & a_2 \\ c_1 & 1 & 1 \\ c_2 & 2 & 2 \end{array} \tag{9}$$

すなわち，図 5 で示した各要素（$a_1$～$a_3$，$b_1$～$b_3$）間に関連がある場合には行列の成分値を 1 に，関連性がない場

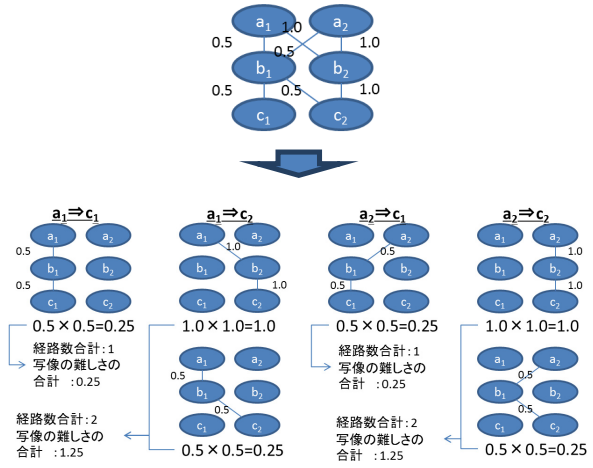



**図 5** 写像経路数と各経路の写像の難しさの算出方法

**Fig. 5** Number of mapping pathway and difficulty of mapping pathway.

合には行列の成分値を 0 に設定し，行列の積をとることとする．積として得られた行列の成分値は，要素間を結ぶ写像経路数を示す．$a_1$ は $b_1$，$b_2$ と，$a_2$ は $b_1$，$b_2$ と関連がある．したがって，式 (9) で要素 a と b の関連を示した行列の成分値はすべて 1 を設定した．同様な考え方から b と c の関連も行列の成分値として示した．その結果得られた行列の解では，$a_1$ と $c_1$ の成分値は 1，$a_1$ と $c_2$ は 2，$a_2$ と $c_1$ は 1，$a_2$ と $c_2$ は 2 となっている．これは図 5 で示した写像経路数とも一致する．

次に，写像の難しさの算出式を示す（式 (10)）．

| | $b_1$ | $b_2$ |
|---|---|---|
| $c_1$ | 0.5 | 0 |
| $c_2$ | 0.5 | 1 |

$\times$

| | $a_1$ | $a_2$ |
|---|---|---|
| $b_1$ | 0.5 | 0.5 |
| $b_2$ | 1 | 1 |

$=$

| | $a_1$ | $a_2$ |
|---|---|---|
| $c_1$ | 0.25 | 0.25 |
| $c_2$ | 1.25 | 1.25 |

(10)

すなわち，行列の成分値に 2 つの要素間の写像の難しさを設定すれば，行列の積で得られる成分値は複数要素間を経由した写像経路の難しさを示す．つまり，上記と同様に $a_1$ と $b_1$ 写像の難しさは 0.5，$a_1$ と $b_2$ とは 1.0，$a_2$ と $b_1$ は 0.5，$a_2$ と $b_2$ は 1.0 を行列の成分値として設定した．要素 b と c についても同様である．その結果得られた行列の解では，$a_1$ と $c_1$ の成分値は 0.25，$a_1$ と $c_2$ は 1.25，$a_2$ と $c_1$ は 0.25，$a_2$ と $c_2$ は 1.25 となっている．これは図 5 で示した写像の難しさとも一致する．

### 3.2 冗長な写像経路数の算出方法

冗長な写像経路数を求める方法について以下に述べる．前節で求めた写像経路数を示す行列の積と単位行列の差が冗長な写像経路数を表している．単位行列は冗長性のない状態であるので，得られた行列から引き，その行列の成分値の総和をとれば冗長経路数の合計値となる．たとえば図 5 では，次式（式 (11)）となる．

| | $a_1$ | $a_2$ |
|---|---|---|
| $c_1$ | 1 | 1 |
| $c_2$ | 2 | 2 |

$-$

| | | |
|---|---|---|
| | 1 | 0 |
| | 0 | 1 |

$=$

| | $a_1$ | $a_2$ |
|---|---|---|
| $c_1$ | 0 | 1 |
| $c_2$ | 2 | 1 |

(11)

すなわち，第 1 項には図 5 における冗長経路数として式 (9) の解を設定し，第 2 項には単位行列を設定している．この単位行列とは，冗長性がない状態，つまり，$a_1$ から $b_1$，そして $c_1$ へ写像される経路，および $a_2$ から $b_2$，そして $c_2$ へ写像される経路を示している．そして，式 (11) より得られた解の成分値の合計値から，図 5 で示したモデルの冗長経路数は 4 本となる．

### 3.3 写像経路の難しさの設定方法

前節までは写像経路の本数とその経路の写像の難しさの算出方法を述べた．しかし，その写像の難しさをそもそもどのように数値化するのか，その算出方法はまだ述べられていない．したがって，本節では，個々の要素の写像の難しさから要素間の写像経路の難しさの算出方法を説明する．

まず，前述（表 1）の特性を考慮し，1 を基準として各要素の持つ写像の難しさを評価し定量化する．基準値 1 に対して，値の幅を等間隔とするため，各要素の定量化にあたっては 0 < 写像の難しさ < 2 とする．ただし，要素間に相互依存がない場合は 0 である．また，1 以上は写像が難しく，1 以下は写像が容易であるものとする．そして，各要素の持つ難しさの相乗平均をその要素間の写像経路に対する写像の難しさとする．したがって，相乗平均をとった値は，式 (10) で示したように，各行列成分値として各行列に設定され，プロジェクトの写像の難しさを求めることに利用される．各要素の持つ写像の難しさの相乗平均値を成分値とした行列の積を行列 A とし，プロジェクトの写像の難しさを式 (12) で定義する（$a_{ij}$ は行列 A の成分値）．

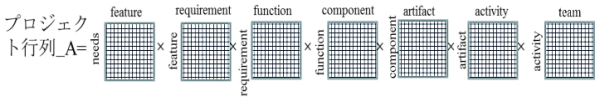
$$\text{冗長化したプロジェクトの写像の難しさ} = |\mathrm{A}| = \sum a_{ij} \tag{12}$$

### 3.4 写像の難しさと冗長性の算出方法の補足とまとめ

前節までは，簡易モデルとして示した図 5 を基に式 (9)，(10) を示した．しかし，実際の開発プロジェクトでは図 1 に示したようなモデルとなるため，本節では実際の開発プロジェクトにおける写像経路数および写像の難しさの算出方法を説明する．その際，図 1 に示した基本パターンの算出方法では対処できない 2 つの派生パターンがある．本節では，その 2 つの派生パターンの算出方法も説明する．

#### 3.4.1 基本パターン

実際の開発プロジェクトでは，式 (9)，(10) に該当するものとして式 (13) を用いる．




(13)

すなわち，図 1 で示したプロジェクトの基本パターンとなる各要素を行または列として作成したマトリックスで構成されている．左から，Needs と Feature 間での写像を第 1 項で示し，次に Feature と Requirement の関連を行列で表現した．同様に Requirement と Function，Function と Component，Component と Artifact，Artifact と Activity，そして最後に Activity と Team の写像関係を行列で示した．そのマトリックスに前述のとおり 1 または 0 を設定し，これら 7 つの行列の積をとれば，写像経路数を求められる．また同様に各要素の写像の難しさの相乗平均値を設定し，これら 7 つの行列の積をとれば，該当の写像経路の難しさを求められる．

#### 3.4.2 派生パターン 1

次に，派生パターンの 1 つ目を説明する．式 (13) では

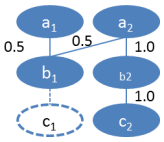



図 6 （例）最下層の要素まで写像経路がないモデル
**Fig. 6** Sample model for lack of mapping pathway to bottom layer.

同一カテゴリどうし（たとえば Team 内の要素間）の写像を表現できないため，数式の変更が必要となる．その場合は，行と列を同一カテゴリとしてその関係性を行列化し，式 (13) に挿入して同様な計算を行えばよい．

### 3.4.3 派生パターン 2

派生パターンの 2 つ目として，**図 6** に示したように最下層の要素まで写像経路が続いていないモデルも式 (13) では表現しきれないため，数式の変更が必要となる．このようなモデルでは，変更しないまま式 (13) を用いると写像経路数も写像の難しさも誤った値が算出されてしまう．

具体的には，この例では本来 $a_1$ から $b_1$ を経由し，再度 $a_2$ に戻って $b_2$，$c_2$ とたどる写像経路を示したい．しかし，図中に示した値を各写像経路における写像の難しさとすると，式 (14) のようになる．

$$\begin{array}{c|cc} & b_1 & b_2 \\ c_1 & 0 & 0 \\ c_2 & 0 & 1 \end{array} \times \begin{array}{c|cc} & a_1 & a_2 \\ b_1 & 0.5 & 0.5 \\ b_2 & 0 & 1 \end{array} = \begin{array}{c|cc} & a_1 & a_2 \\ c_1 & 0 & 0 \\ c_2 & 0 & 1 \end{array} \tag{14}$$

すなわち，得られた行列は $a_2$ と $c_2$ の関連を示した成分以外はすべて 0 の値となっている．式 (14) の 1 項目には，要素 b と c の関連が示されている．関連を持つのは $b_2$ と $c_2$ のみである．その写像の難しさを行列成分として設定し，他の関連は成分値 0 として設定されている．また 2 項目には要素 a と b との関連が示されている．関連を持っていないのは $a_1$ と $b_2$ のみであり，行列成分を 0 としている．それ以外は図中に示した写像の難しさを行列の成分値とした設定している．この結果得られた行列の積は，$a_2$–$c_2$ の成分のみ 1 を示し，他成分は 0 となっている．つまり，モデル全体の写像の難しさは，$a_2$ から $b_2$ を経由して $c_2$ までの写像経路の難しさのみで決まっており，他の $a_1$ や $b_1$ の要素の持つ写像の難しさがモデル全体の難しさに反映されていない．そこで，

$$\begin{array}{c|cc} & b_1 & b_2 \\ c_1 & 0 & 0 \\ c_2 & 0 & 1 \end{array} \times \begin{array}{c|cc} & a_1 & a_2 \\ b_1 & 0.5 & 0.5 \\ b_2 & 0 & 1 \end{array} \times \begin{array}{c|cc} & b_1 & b_2 \\ a_1 & 0.5 & 0 \\ a_2 & 0.5 & 1 \end{array}$$

$$\times \begin{array}{c|cc} & a_1 & a_2 \\ b_1 & 0.5 & 0.5 \\ b_2 & 0 & 1 \end{array} = \begin{array}{c|cc} & a_1 & a_2 \\ c_1 & 0 & 0 \\ c_2 & 0.25 & 1.25 \end{array} \tag{15}$$

のように，a と b の関係を示す転置行列を式 (14) に挿入することによって，写像経路を数式化する．その結果，$a_1$ から $c_2$ の写像経路の難しさも正しく反映されるようになる．式 (14) では要素 a から c へ，つまり上から下に向けた写像経路しか表現されない．そのため計算上，$b_1$ と $c_1$ の関連がなければ，$a_1$ から $c_1$ への写像経路はないものされてしまう．$a_2$ から $b_1$ への経路も，$b_1$ から $c_1$ への経路が存在しないために成分値は 0 となる．そこで，a と b の関係を示す転置行列を挿入することで（式 (15) の左から第 3 項），b から a に向けた下から上に向けた写像経路を行列計算の中に組み込んだのである．これにともなって，転置行列によって b から a に向けた写像を再度 a から b に向けた写像にするために式 (15) の左から第 2 項の行列を挿入している．

同様に写像経路数の算出式は，次式（式 (16)）となる．

$$\begin{array}{c|cc} & b_1 & b_2 \\ c_1 & 0 & 0 \\ c_2 & 0 & 1 \end{array} \times \begin{array}{c|cc} & a_1 & a_2 \\ b_1 & 1 & 1 \\ b_2 & 0 & 1 \end{array} \times \begin{array}{c|cc} & b_1 & b_2 \\ a_1 & 1 & 0 \\ a_2 & 1 & 1 \end{array}$$

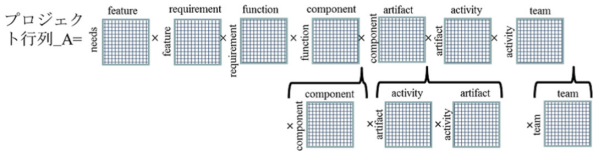
$$\times \begin{array}{c|cc} & a_1 & a_2 \\ b_1 & 1 & 1 \\ b_2 & 0 & 1 \end{array} = \begin{array}{c|cc} & a_1 & a_2 \\ c_1 & 0 & 0 \\ c_2 & 1 & 2 \end{array} \tag{16}$$

すなわち，式 (15) における成分値を 0 または 1 に置き換えた行列で構成したものとした．ここで示したように $a_1$ から $c_2$ は 1 つの経路のみであり（行列の $a_1$–$c_2$ の成分値），式 (15) では成分値が $0.5 \times 0.5 \times 1.0 \times 1.0 = 0.25$ となっている．また，$a_2$ から $c_2$ については転置行列を挿入したことで 2 つの経路がある．1 つは $a_2$ から $b_2$ を経由して $c_2$ をたどる経路である．その場合の計算式は $1.0 \times 1.0 = 1.0$ となる．もう 1 つは $a_2$ から $b_1$ を経由して $a_2$ に戻り $b_2$ を経由して $c_2$ をたどる経路である．この場合は $0.5 \times 0.5 \times 1.0 \times 1.0 = 0.25$ となる．これらを合計した結果 $a_2$ から $c_2$ への経路の成分値は 1.25 のとなる．

### 3.4.4 まとめ

以上のように基本パターンとして式 (13) を示し，その派生パターンを 2 つ説明した．一般的にここで説明したパターンを用いればプロジェクトの写像経路数や写像の難しさを算出することができると考える．

本研究の実証検証で用いたプロジェクトにおいても，




(17)

のように，Team 間の関連や Component 間の関連を表現するために 1 つ目の派生パターンを用い，企画書等の Artifact を介した Activity との写像経路を表現するためには 2 つ目の派生パターンを用いた．以上をまとめると冗長化したプ

ロジェクトの写像の難しさは式 (12) で示され，また，

$$\text{並行経路数} = \text{行列の次数} \tag{18}$$

$$\text{冗長な経路数} = |\mathrm{A}' - \mathrm{E}| = \sum |a'_{ij} - e_{ij}| \tag{19}$$

となる．ただし，行列 A′ は行列 A の成分値を 0，または 1 とした行列であり，E は単位行列である．$a_{ij}$ または $a'_{ij}$ は行列 A および A′ の成分値を示す．

## 4. ソフトウェア開発プロジェクトの複雑性指標の妥当性検証

実際のゲームソフトウェア開発を対象にそのプロジェクトの複雑性の妥当性検証を行った．妥当性検証にあたっては，ゲームソフトウェア開発プロジェクトをモデル化し，その複雑性と生産性および開発工数との相関を検証した．

### 4.1 検証プロジェクトの選定とプロジェクトの概要

まず，妥当性検証を行うにはどのようなプロジェクトを選定するべきか，その条件を検討した．そして，① 相関検証するうえで十分なデータ数を得られること，② プロジェクトを構成する個々の要素の持つ写像の難しさを高い信頼性で数値化できること，そして ③ 複数の開発者がコミュニケーションをとりながら開発プロセスを進め，複数の要素で構成されるプロダクトを生み出していること，以上の 3 つの条件を満たすこととした．

そして，これらの条件を満たすプロジェクトとして，日本国内のあるゲーム開発プロジェクトを対象とすることとした．本プロジェクトの開発は 6–7 名で行われており，ゲームソフトウェアという複雑な機能を有する開発を進めていた．また，対象としたのは 21 のプロジェクトであり，相関を検証するうえでは十分なデータ数と考えた．さらに，この 21 のプロジェクトのプロジェクトマネージャは 1 人であり，(詳細は後述するが) プロジェクトを構成する個々の要素の持つ写像の難しさを高い信頼性で数値化できると考えた．以上のようにはじめにあげた 3 つの条件を満たしていたことから，本プロジェクトで検証を行うこととした．

プロジェクトの開発期間は 2010 年 12 月から 2011 年 8 月である．この間，プロジェクトでは要求仕様の整理から基本設計，実装，試験という一連の開発工程を実施している．7 プロジェクトごとに開発を行い，顧客への納品と検収を繰り返して 21 プロジェクトの開発を完了した．また，個々のプロジェクトの開発完了の判断は，バグ発生状況が収束傾向にあり，かつバグ抽出件数がある一定値以上に達することを条件とした．なお，今回対象とした 21 プロジェクトのうち，No.7，14，21 のプロジェクトは**図 7** の中で Activity としてのアダプタ作成，アダプタ（Artifact）およびアダプタ（Component）の要素が存在するモデルであるが，他の 18 プロジェクトではこれらの要素は存在しない．また同様に，No.7，14，21 のプロジェクトでは Team におけるサウンドデザイナは存在しないが，他の 18 プロジェクトではこの要素は存在するモデルとなる．

### 4.2 検証対象プロジェクトのモデル化

対象としたゲーム開発プロジェクトの構成要素を抽出する．プロセス領域・実態領域・機能領域・顧客領域においてどのような要素から構成されているのか把握するために，プロジェクトのディレクタ（プロジェクトマネージャの役割）からヒアリングを行った．その結果を図 7 にまとめた．そして，他のソフトウェア開発プロジェクトと非常に類似した要素で構成されていることが分かった．したがって，本プロジェクトが他にも応用可能な検証プロジェクトであると考える．また，図 7 に示した情報写像モデルは，四角の箱内に記載された要素名が，ヒアリングにより抽出したプロジェクトを構成する要素であり，要素間を結んだ実線は写像経路を示している．なお，本図は UML ツールを用いて作成したが，必ずしも同ツールを用いる必要はない．

#### 4.2.1 プロセス領域の構成要素

まず，プロセス領域の Team の要素を整理するために，組織体制についてディレクタからヒアリングを行った．

プロジェクトマネージャとしてゲーム開発ではディレクタ，ビジネスアナリストとしてプランナ，またアーキテクトとしてチーフプログラマがおり，そしてプログラマが存在する．またユーザインタフェース（以下 UI と記述）デザイナはサウンドデザイナ，グラフィッカ等が該当する．このようにプロジェクトの体制を調査し，Team の要素を定義し，図 7 に示した．

次に Team 内の写像経路について述べる．Team 内での写像経路は設計に直接関わらない写像を示す．つまり Activity によって Artifact を作成するための写像以外を示す．たとえば，企画書をまとめるために企画書作成という Activity を介した写像経路がある．企画書という Artifact を生成するので，この写像経路は設計作業に直接関連する写像といえる．一方，Activity を介さない Team 内に閉じた写像経路は，たとえば進捗管理等の管理上の写像経路を示す．進捗状況の共有は Artifact を生成する直接的な写像ではない．しかし，間接的には個々の Team の設計作業にも影響を及ぼすため，写像経路としてモデルに組み込むこととした．

次に個々の要素間の関連について述べる．図中に示した Team の要素のうち，大手メーカプロデューサ以外の要素が実際にソフトウェアを開発する会社の役割を示している．プロジェクトマネージャの役割はディレクタなので，プランナ，サウンドデザイナ，チーフプログラマは各自の作業状況をディレクタと共有している．エンジンプログラマについては，エンジン自体が大手メーカから提供されていたため，大手メーカプロデューサとの関連付けがある．

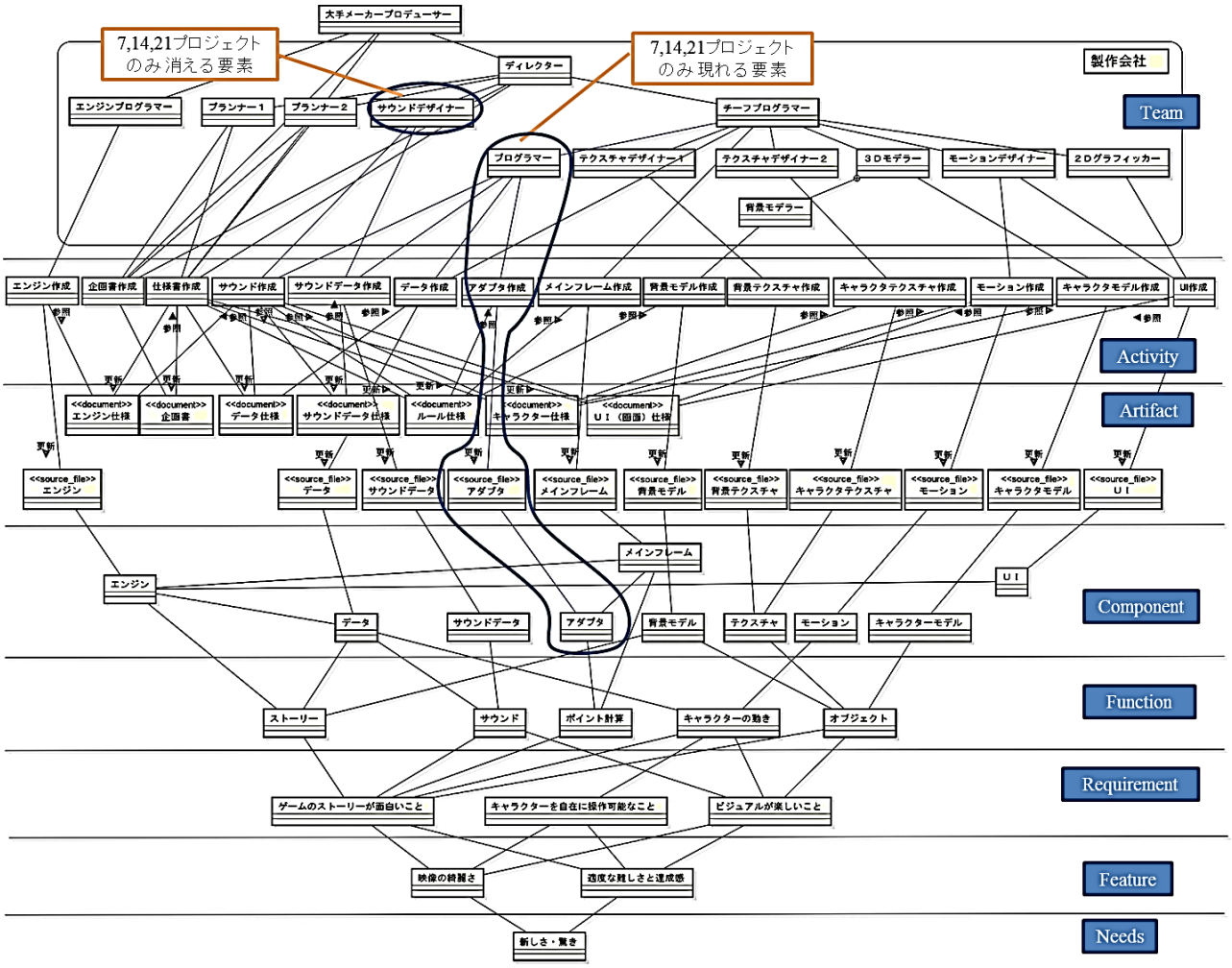



図 7　実証検証を行ったプロジェクトの情報写像モデル
**Fig. 7**　Information mapping model of validation project.

また，チーフプログラマはプログラマやゲームに登場するキャラクタやそのルール等を開発するための 3D モデラ，背景モデラ，モーションデザイナ，テクスチャ，2D グラフィッカを率いていた．

なお，アダプタ開発を行うプロジェクトは 7 名の開発要員で開発し，他のプロジェクトは 6 名の開発要員で開発している．要員数の内訳は以下のとおりである．ディレクタはプランナの役割も兼任して 1 名，またもう 1 名のプランナ，サウンドデザイナ，プログラマの役割は各 1 名で構成した．また背景モデラと背景テクスチャデザイナ（図中，テクスチャ 1）は兼任で 1 名，3D モデラ，キャラクタテクスチャデザイナ，2D グラフィッカ，モーションデザイナも兼任で 1 名，エンジンプログラマ，チーフプログラマは兼任で 1 名である．

次に，プロセス領域の Activity の要素を整理するために，開発プロセスをヒアリングした．基本要件を決め，要求仕様書，設計書を作成し，それを基に実装していくというプロセスは，組織体制と同様に他のソフトウェア開発にも非常に類似していた．そのプロセスを調査し，Activity の要素を定義し，図 7 に示した．

上記の Activity に関する留意点は “試験” に関連した要素としてあげていないことである．なぜならば，公理的設計論では，写像の方向によって，設計過程とその試験課程を表現しているからである．したがって，開発プロセスモデルである V 字モデル [26] の左側の作業のみをあげている．

Activity の写像経路について述べるが，Activity どうしの写像経路は存在しない．Team の各要素が行う Activity を実線で結んでいる．またその Activity から生成される Artifact を実線で結び，その写像経路を整理した．

次に，同様に，成果物を調査し，Artifact を定義し，図 7 に示した．図中に示した要素名の上に document と記述したものは日本語や図表等で作成された仕様書・設計書である．同様に source_file と記述した要素はプログラミング言語で作成されたコンパイル前の電子ファイル等を示す．

また，要素間の関連については，Artifact どうしの関連はない．したがって，上下層との関連のみを実線で示した．企画書作成（Activity）で作成された企画書を別の仕様書作成（Activity）が参照し，エンジン仕様，データ仕様，ルー

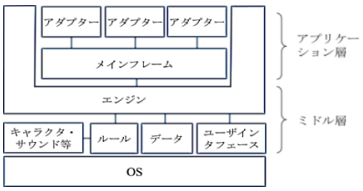



図 8 ゲームソフトウェアの構造
**Fig. 8** Game software architecture.

ル仕様，キャラクタ仕様，UI 仕様を作成していた．サウンドデータ仕様はエンジン仕様やデータ仕様をもとにサウンド作成（Activity）を介して作成されている．また，これらの仕様書をもとにして上記であげた各 source_file が作成される．

#### 4.2.2 実体領域の構成要素

実体領域を構成する要素を明らかにするために，ゲームソフトウェアのアーキテクチャを調査し，どのような Component と Function で構成されているのか調査した．

ゲームのソフトウェア構造は，他の企業向けソフトウェアと類似したソフトウェアアーキテクチャである（**図 8**）．

その結果をもとに，Component と Function の要素とその関連を定義し，図 7 に示した．なお，本プロジェクトでは，図 8 で示した要素の 1 つであるルールは関連する要素（背景モデル，テクスチャ，モーション，キャラクタモデル，サウンドデータ）の一部として位置付けていた．したがって，下記の要素にはあげていないが，各要素には包含されている．また，キャラクタやサウンドデータ等は，ルールによりエンジンとは独立性の高い部品となっていた．そのため，図 7 ではエンジンと関連付けていない．

#### 4.2.3 機能領域と顧客領域の構成要素

機能領域と顧客領域の構成要素を整理するために，ゲームソフトウェアに対する顧客の要求がどのようなものなのか調査した．

顧客の要求も他のソフトウェア開発プロジェクトとも類似した構造によって規定されている．ゲーム開発において，ユーザは，新しさと驚きを感じられるような映像とロールプレイングをゲームで体感すること（新しさ・驚き）を望んでいる（Needs）．その実現のために，基本要件（Feature）として，ゲームソフトウェアは，① ストーリに即した色彩的な綺麗さと，高い解像度を活かしたデザインを実現したものであること（映像の綺麗さ）と，② ロールプレイ上で，戦略的な駆け引きを必要とした多様なストーリを実装していること（適度な難しさと達成感）が規定された．したがって，要素間については，1 つの Needs から 2 つの Feature に関連付けを行った．

また，それを実現するために各機能に求められる要求条件（requirement）として，ゲームソフトウェアは，① 提供する UI によって，キャラクタの動くスピード，方向を自在に操作可能とするための機能を具備していること（キャラクタを自在に操作可能なこと），② 提供するストーリ展開において，サウンド効果を用いて臨場感を持たせ，キャラクタがシステムと駆け引きすることを可能にする．そして，その結果獲得されるポイントによって，ミッションの達成度を判定する機能を具備していること（ゲームのストーリが面白いこと），③ その映像デザイン・キャラクタの動き・サウンド，これら 3 つの要素を自在に組み合わせ，コントロールすることで，映像デザインからの視覚効果を補強する機能を具備していること（ビジュアルの楽しさ）が抽出された．これらの関連を図 7 に示した．Needs と Feature を顧客領域の要素とし，Requirement は機能要求に等しいので機能領域の要素とした．なお，秘匿義務があるため，各要素の記述は上記のレベルにとどめた．

#### 4.2.4 構成要素の粒度に関する補足

前述のように情報写像モデルの構成要素を決定した．本項では，なぜ，構成要素をこの粒度にしたのか説明する．

情報写像モデルを構成する要素の粒度は，そのモデル作成および管理工数とプロジェクトに求められる開発コストの管理精度の間で，そのトレードオフを考慮して決定されるべきである．ただし，モデルの可読性を考慮すると，筆者らの経験上，各カテゴリの要素数は 10–20 程度にとどめることが適切と考える．したがって，それ以上の詳細なモデルを作成する場合は，サブシステム分割により，個別にモデル化することが望ましい．その際，プロジェクト全体の複雑性を算出するためには，その全体を表す行列をブロック（小行列）化し，対角上にあるブロックを各サブシステムに割り当てる．非対角成分ではサブシステム間の写像経路を表現する．このように 1 つの行列をブロック化することで，プロジェクト全体の複雑性も算出することが可能である [19]．

### 4.3 検証データの収集

#### 4.3.1 複雑性指標の算出

複雑性指標を算出するため，写像経路数と要素の持つ写像の難しさを以下のように規定し，複雑性を算出した．

本検証では，1 人のディレクタが 21 個のプロジェクトそれぞれの開発状況を把握している．まず，抽出した要素で構成する行列の次数により並行経路数を決定し，全写像経路数はプロジェクトの情報写像モデル（図 7）より規定した．そして，プロジェクトを構成する各要素の持つ写像の難しさについては 1 人のディレクタに表 1 をガイドラインとしてヒアリングし設定した．その結果は**表 2** のとおりである．この結果をもとに，3.3 節で述べた方法により，写像経路の難しさを規定した．これら写像経路数と写像の難しさを規定するにあたっては，3 章で述べた計算式に則り，表計算シートによってプロジェクトごとにその複雑性を算出した．なお，本計算方法では，要素の持つ写像の難しさの微妙な差が，最終結果で増幅されるようなことはない．

表 2 実証検証を行ったプロジェクトの要素とその写像の難しさ
**Table 2** Difficulty of mapping for project elements.

| | | プロジェクト | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| team | 大手メーカープロデューサー | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | エンジンプログラマー | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | ディレクター | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.2 | 0.2 | 0.2 | 0.2 | 0.1 | 0.1 | 0.1 |
| | プランナー1 | 1.9 | 1.9 | 1.9 | 1.8 | 1.7 | 1.6 | 1.9 | 1.5 | 1.5 | 1.4 | 1.3 | 1.2 | 1.1 | 1.3 | 1.2 | 1.1 | 1 | 1 | 1 | 1 | 1 |
| | プランナー2 | 1.9 | 1.9 | 1.9 | 1.8 | 1.7 | 1.6 | 1.9 | 1.5 | 1.5 | 1.4 | 1.3 | 1.2 | 1.1 | 1.4 | 1.3 | 1.3 | 1.3 | 1.2 | 1.2 | 1.2 | 1.1 |
| | チーフプログラマー | 0.7 | 0.7 | 0.6 | 0.6 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.4 | 0.4 | 0.5 | 0.3 | 0.3 | 0.3 | 0.3 | 0.2 | 0.2 | 0.1 |
| | プログラマー | – | – | – | – | – | – | 1.2 | – | – | – | – | – | – | 1 | – | – | – | – | – | – | 1.5 |
| | テクスチャデザイナー1 | 1.1 | 1.1 | 1.1 | 1.1 | 1 | 1 | 1 | 0.8 | 0.8 | 0.7 | 0.7 | 0.6 | 0.6 | 0.5 | 0.3 | 0.3 | 0.3 | 0.2 | 0.2 | 0.1 | 0.1 |
| | テクスチャデザイナー2 | 1.1 | 1.1 | 1.1 | 1 | 1 | 1 | 1 | 0.9 | 0.9 | 0.7 | 0.7 | 0.6 | 0.5 | 0.5 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.2 |
| | モーションデザイナー | 0.6 | 0.6 | 0.6 | 0.5 | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.3 | 0.3 | 0.3 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| | 2Dグラフィッカー | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.8 | 0.8 | 0.7 | 0.6 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.4 | 0.4 | 0.4 | 0.4 |
| | 3Dモデラー | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.7 | 0.7 | 0.6 | 0.5 | 0.5 | 0.4 | 0.4 |
| | 背景モデラー | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.8 | 0.7 | 0.7 | 0.6 | 0.5 | 0.5 | 0.5 |
| | サウンドデザイナー | 1.7 | 1.7 | 1.6 | 1.6 | 1.5 | 1.5 | – | 1.4 | 1.4 | 1.3 | 1 | 1 | 1 | – | 1 | 1 | 1 | 1 | 1 | 1 | – |
| activity | エンジン作成 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 |
| | 企画書作成 | 1.1 | 1.1 | 1 | 0.9 | 0.9 | 1 | 0.8 | 0.8 | 0.8 | 0.8 | 0.7 | 0.6 | 0.8 | 1 | 0.8 | 0.8 | 0.9 | 0.7 | 0.7 | 0.8 | 0.6 |
| | 仕様書作成 | 1.7 | 1.7 | 1.3 | 1.5 | 1.3 | 1.5 | 1.3 | 1.1 | 1.1 | 1.4 | 1.4 | 1.7 | 1.4 | 1.2 | 1.1 | 1.1 | 0.7 | 1.4 | 0.7 | 1.1 | 1.1 |
| | データ作成 | 1.9 | 1.6 | 1.2 | 1.9 | 1.6 | 1.9 | 0.7 | 1.6 | 1.2 | 1.2 | 1.6 | 1.9 | 1.6 | 0.9 | 0.8 | 0.8 | 0.8 | 1.2 | 0.8 | 0.8 | 0.5 |
| | アダプタ作成 | – | – | – | – | – | – | 1.9 | – | – | – | – | – | – | 1.2 | – | – | – | – | – | – | 1.9 |
| | メインフレーム作成 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 |
| | 背景モデル作成 | 1.9 | 1.9 | 1.2 | 1.6 | 1.2 | 1.2 | 0.8 | 1.2 | 1.2 | 1.5 | 1.6 | 1.9 | 1.6 | 1.6 | 1.2 | 1.3 | 1.2 | 1.6 | 1.2 | 1.6 | 0.1 |
| | キャラクタモデル作成 | 1.9 | 1.9 | 1.6 | 1.9 | 1.6 | 1.6 | 0.9 | 1.5 | 1.7 | 1.6 | 1.7 | 1.7 | 1.9 | 1.2 | 1.2 | 1.2 | 1.5 | 1.3 | 1.3 | 1.5 | 0.4 |
| | 背景テクスチャ作成 | 1.3 | 1.5 | 1 | 1.3 | 1.1 | 0.9 | 0.6 | 0.9 | 1 | 1.1 | 1.1 | 1.2 | 1.3 | 0.9 | 0.8 | 0.9 | 0.9 | 1.2 | 0.9 | 1 | 0.1 |
| | キャラクタテクスチャ作成 | 1.9 | 1.9 | 1.2 | 1.7 | 1.6 | 1.8 | 0.8 | 1.5 | 1.6 | 1.5 | 1.7 | 1.5 | 1.9 | 1.4 | 1.2 | 1.2 | 1.3 | 1.6 | 1.2 | 1.6 | 0.4 |
| | モーション作成 | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 | 0.7 | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.8 |
| | UI作成 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.5 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.6 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.4 |
| | サウンドデータ作成 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 |
| | サウンド作成 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.1 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.1 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.1 |
| artifact | 企画書 | 0.5 | 0.5 | 0.4 | 0.3 | 0.3 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| | エンジン仕様 | 1.2 | 1.2 | 1.1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | データ仕様 | 1.1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | ルール仕様 | 1.7 | 1.7 | 1.6 | 1.6 | 1.5 | 1.5 | 1.5 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 |
| | キャラクター仕様 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | UI(画面)仕様 | 0.4 | 0.4 | 0.3 | 0.3 | 0.3 | 0.3 | 0.6 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.5 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.5 |
| | サウンドデータ仕様 | 0.4 | 0.4 | 0.3 | 0.3 | 0.3 | 0.3 | 0.5 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.5 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.5 |
| | エンジン(ソース) | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 |
| | データ(ソース) | 0.7 | 0.7 | 0.6 | 0.6 | 0.6 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 |
| | アダプタ(ソース) | – | – | – | – | – | – | 1.9 | – | – | – | – | – | – | 1.9 | – | – | – | – | – | – | 1.9 |
| | メインフレーム(ソース) | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 |
| | 背景モデル(ソース) | 1.6 | 1.6 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 |
| | 背景テクスチャ(ソース) | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 |
| | キャラクタモデル(ソース) | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 |
| | キャラクタテクスチャ(ソース) | 1.6 | 1.6 | 1.5 | 1.5 | 1.5 | 1.5 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 | 1.4 |
| | モーション(ソース) | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.6 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.5 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.5 |
| | UI(ソース) | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| | サウンドデータ(ソース) | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.4 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| component | エンジン | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 | 1.7 |
| | データ | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| | アダプタ | – | – | – | – | – | – | 1.5 | – | – | – | – | – | – | 1.5 | – | – | – | – | – | – | 1.5 |
| | メインフレーム | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 |
| | 背景モデル | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1 |
| | テクスチャ | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| | モーション | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 1 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 1 |
| | キャラクターモデル | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| | UI | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | サウンドデータ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| function | ストーリー | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 0.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 0.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 0.5 |
| | ポイント計算 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| | キャラクターの動き | 1 | 1 | 1 | 1 | 1 | 1 | 1.3 | 1 | 1 | 1 | 1 | 1 | 1 | 1.3 | 1 | 1 | 1 | 1 | 1 | 1 | 1.2 |
| | オブジェクト | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 0.6 |
| | サウンド | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| requirement | キャラクタを自在に操作可能 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | ゲームのストーリーが面白い | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.5 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| | ビジュアルの楽しさ | 1 | 1 | 1 | 1 | 1 | 1 | 0.5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.5 |
| feature | 映像の綺麗さ | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| | 適度な難しさと達成感 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| needs | 新しさ・驚き | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 | 1.9 |

以下，表 2 で示した各要素の写像の難しさについて述べる．表中で数値を記入していない箇所はその要素が存在していないことを示している．たとえば 7，14，21 以外のプロジェクトではアダプタ作成（Activity）に数値が記入されていないのは前述のとおり，プロジェクトで作業がなかったからである．また各要素の傾向については，Team，Activity，Artifact の要素の傾向としては制作が進むにつれて習熟度も向上することから値が小さくなっていく傾向がみられる．しかし，Component や Function についてはプロジェクトによって数値の大小があるが，それはプロジェクトの仕様の違いから生じているものであった．Requirement，Feature，Needs についてはアダプタ開発の有無によって違いはあるものの，その他のプロジェクトはまったく同じ値を示している．

#### 4.3.2 開発工数データの収集と平均作業日数の算出

プロジェクトごとに各作業の稼動工数データを収集した．収集した稼動工数をプロジェクト単位に合計し，それを開発工数とした．また開発工数を各プロジェクトの要員数で割ることによって 1 人あたりの平均作業日数を算出した．なお，大手メーカプロデューサは発注側であるため，開発要員数および開発工数のデータには含めていない．

#### 4.3.3 収集データの信頼性向上のための方策

写像の難しさの数値化は前述のとおり 1 人のディレクタにより実施した．なぜならば，複数人で設定値を決める場合，その設定基準の認識が完全に合わない限り，設定値の大小関係を可能な限り正確に得ることが困難だからである．そこで本研究では，1 人の設定者が複数の要素を相対評価することによって，設定者が異なることによる主観の違いから生まれるノイズを排除した．したがって，21 個のプロジェクトを熟知し評価可能である 1 人のディレクタによって，その数値設定を行うこととした．

### 4.4 実証検証

#### 4.4.1 検証の進め方

まず，複雑性と 1 人あたりの平均作業日数の相関を確認する．式 (2) の検証により式 (1) および (3) の妥当性を確認する．

次に，複雑性と工数の相関を確認する．式 (8′) を検証することで式 (8) の妥当性を確認する．式 (3) から式 (5) を算出し，その結果をもとに式 (4) を求める．その結果から式 (8) の計算を行い，式 (8′) の検証を行う．

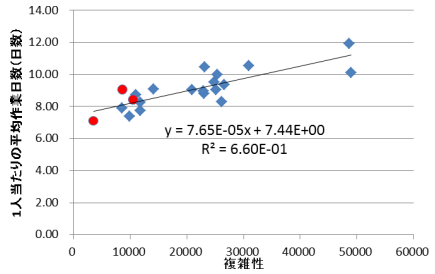



図 9 複雑性と 1 人あたりの平均作業日数の相関

**Fig. 9** Graph relating complexity to average work days per person.

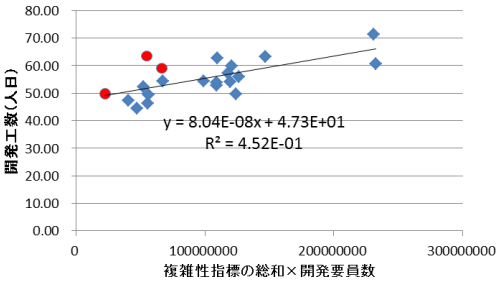



図 10 複雑性指標の総和と開発工数の相関

**Fig. 10** Graph relating total complexity to man-hour.

#### 4.4.2 複雑性と生産性（1 人あたりの平均作業日数）の相関

複雑性を x 軸，1 人あたりの平均作業日数を y 軸として，検証の結果を以下に示す．

すなわち，**図 9** により，プロジェクトの複雑性は 1 人あたりの平均作業日数と非常によく相関していることが分かる．近似式は図に示したとおりであり，相関係数は 0.813（有意確率 $< 0.001$）である．

以上のように，提案した複雑性指標と 1 人あたりの平均作業日数には強い相関がみられた．図中の 3 つの丸印のデータはアダプタ開発を行ったプロジェクトであり，他の 18 個の四角印のデータはそれ以外のサウンドデザイナがいないプロジェクトのデータである．プロジェクトの体制が異なるにもかかわらず，相関係数が示しているとおりに 21 のプロジェクトが非常によく近似直線上に乗っていることが分かる．以上により，プロジェクトの複雑性指標がプロジェクトの生産性をマネジメントするうえで有用であることを検証した．

#### 4.4.3 複雑性指標の総和と開発要員数の積と開発工数の相関

複雑性指標の総和と開発要員数の積を x 軸，開発工数を y 軸として，検証結果を以下に示す．

すなわち，**図 10** より，複雑性指標の総和と開発要員数の積と，開発工数は強く相関（相関係数は 0.672（有意確率 $< 0.001$））していることが分かる．なお，近似線は図示したとおりである．したがって，式 (8′) が確認されたので，式 (8) は成立すると考える．その結果，プロジェクトの複雑性のマネジメントは，開発工数の低減に効果があることが分かり，マネジメント指標として有用であることを確認した．

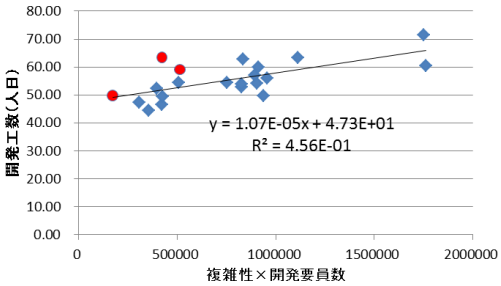



図 11 複雑性と開発工数

**Fig. 11** Graph relating complexity to man-hour.

次に，式 (8) による効果について述べる．2.7 節で述べたように，式 (2) から，複雑性と要員数の積も，開発工数と相関すると考えられている．**図 11** に，複雑性と開発要員数の積を x 軸，開発工数を y 軸として，その相関を検証したグラフを示す．

すなわち，図 11 より，複雑性と要員数の積と，開発工数も強く相関（相関係数は 0.675（有意確率 $< 0.001$））していることが分かる．なお，近似線は図示したとおりである．

図 10 と図 11 のグラフを比較すると，相関係数は 0.672 と 0.675 と大差なく，有意確率も互いに 0.001 以下であった．つまり，式 (8) を用い，経路数に応じた複雑性を積み上げて開発工数を算出しても，プロジェクトに存在する全経路の複雑性から開発工数を算出しても変わりがないこととなる．相関係数が 0.67 を示していることから概算で開発工数を算出するうえでは，どちらの方法でも算出結果に差異はないと考える．しかし，今回の検証では，式 (8) における $\alpha_j$ の精度に課題があると考える．その改善により，今後さらなる相関係数の向上が見込める．その課題とは，式 (8) の導出時に，プロジェクトの全開発期間を通して要員数を一定とし，冗長な経路数を 0 本のときから最大値 $n$ 本まで各 $\alpha_j$ との積を求め，その総和を算出している点である．なぜなら，実際の写像経路数の変化は，たとえば，全要員が関わって 0 本から $n$ 本まで変化したわけではなく（つまり，その間の $\alpha_j$ はゼロ），ある本数（$x$）から $n$ 本までの限定された変化だった可能性が考えられるからである．その場合，0 から $x$ 本までの間に積み上げた複雑性の和は，本来生じていないものであり，誤差の要因となりうる．このように $\alpha_j$ のデータ精度を改善することにより，式 (8) の精度向上が見込めると考えられる．

#### 4.4.4 相関に関する補足データ

本研究で提案した複雑性指標を算出するうえでは，プロジェクトの様々な要素をもとにしている．個々の要素自体も当然，生産性や開発工数に効く要因を持っている．したがって，本項では，複雑性と生産性や開発工数との相関が，ある特定の要素に起因したものなのか否かを確認する．

**表 3** 各要素カテゴリと生産性，または工数との相関する要素数の状況

**Table 3** Assessment of the correlation.

| 要素のカテゴリ | 平均作業日数との相関 | | | 工数との相関 | | |
|---|---|---|---|---|---|---|
| | ○ | △ | × | ○ | △ | × |
| team | 5 | 6 | 3 | 5 | 6 | 3 |
| activity | 7 | 1 | 6 | 1 | 6 | 7 |
| artifact | 6 | 2 | 10 | 4 | 7 | 7 |
| component | 0 | 0 | 10 | 0 | 0 | 10 |
| function | 0 | 0 | 5 | 0 | 0 | 5 |
| requirement | 1 | 0 | 2 | 0 | 1 | 2 |
| feature | 0 | 0 | 2 | 0 | 0 | 2 |
| needs | 0 | 0 | 1 | 0 | 0 | 1 |
| 小計 | 19 | 9 | 39 | 10 | 20 | 37 |
| 合計 | 67 | | | 67 | | |

相関係数の値による各カテゴリの定義
- 強い相関（○）：0.6以上
- 弱い相関（△）：0.4以上0.6未満
- 相関無し（×）：0.4未満

表 2 で示したプロジェクトの個々の要素の持つ写像の難しさと，生産性や工数との相関についてまとめたデータを**表 3** に示す．

すなわち，ある特定の要素が相関要因となっているわけではない．図 7 で示した各要素のカテゴリが同一であっても，相関するか否かは要素次第である．また相関する要素に限定してその特性を調査しても，共通する特定の性質（たとえば多くの経路が通っている等）は発見されなかった．したがって，生産性や工数と相関した指標として，個々の要素の相関を個別に確認し活用するよりも，本研究で示した複雑性指標を用いる方が有効と考える．

## 5. ソフトウェア開発プロジェクトにおける写像経路の影響についての考察

検証の結果，提示したモデルの複雑性と生産性や開発工数と相関することが分かった．その複雑性は写像経路を変数としているが，生産性や開発工数と写像経路の関連について，まだ述べられていない．そこで，本章では，その複雑性を生じさせている写像経路と生産性や開発工数にはどのような関連があるのか考察を行う．

### 5.1 設計プロセスおよびコミュニケーションに及ぼす写像経路の影響分析

一般的には生産性向上や開発工数の低減は，設計情報の写像という観点よりも設計プロセスの観点で考えられることが多い．たとえば，生産性向上のためにソースコードの自動生成ツールを用いることで実装プロセスを省略できる．それにより，プロジェクトの生産性が向上し開発工数が低減されると考えられている．そこで本節では設計プロセスに対して写像経路がどのように影響を及ぼすのか，特にコミュニケーションの観点を含めて，その関連を明らかにする．

設計プロセスにおいて，個々の開発者が設計解を得るために 2 つの手順を踏む．① 設計条件を整理すること．② その条件を満たすための解を複数案検討し，その最善案を選択すること．これらの 2 つの手順が，設計解を得るために必要である．ソフトウェアを構成する部品間で冗長な経路がある場合，① の設計条件を整理する際に互いの条件を擦り合わせること，すなわち，コミュニケーションが必要となる．たとえば，インタフェースすべきデータ項目の調整等がそれにあたる．図 1 にあてはめると，部品（Component）間の連携がある場合には開発者（Team）間で作業（コミュニケーション）を通して部品（Component）間のインタフェースを規定する．その規定した内容は設計書（Artifact）に反映され，最後に部品（Component）にも実装されることと解釈される．このプロセスは，複雑性指標の概念においては要素間の相互依存性を下げること，つまり冗長な経路数を低減させることによって生産性向上や開発工数の低減が可能である．そして，インタフェースすべきデータ項目が決まってしまえば，それ以降の手順は個々の開発者に閉じた作業となる．つまり，② の作業は個々の開発者の作業となり，その作業に開発者間に複数の冗長な写像経路が生じていない．したがって，他者との調整作業（コミュニケーション）をしなくても，個々の開発者の写像能力が高ければ開発コストは下がるのである．このように設計プロセスの観点からも複雑性指標と生産性向上と開発工数低減との相関を理解することができる．特に，冗長な写像経路を低減させることは，設計プロセスとして行われるコミュニケーションの軽減につながることが理解される．

### 5.2 写像経路のマネジメントによる工数低減および生産性向上策の検討

本節では，図 4 で示した生産性，複雑性と冗長な経路数の関係を用いて，生産性の向上や開発工数の低減のための写像経路のマネジメント方法について述べる．

前節で示された図 4 により図の物理的な意味を考察すると，工数低減・生産性向上には，① 1 本平均の写像の難しさを小さくすること，② プロジェクトを構成する要素数を少なくすること，③ 冗長な写像経路を少なくすることが必要であることが有効であることが分かった．なぜならば，生産性向上には，グラフの傾きの低減，y 切片の低減が必要だからである．グラフの傾きの低減には，1 本平均の写像の難しさを小さくすることが必要である．また y 切片の低減には基本プロジェクトの写像の難しさを小さくすることが必要である．それにはグラフの傾きの低減と同様に 1 本平均の写像の難しさを小さくすること，または，並行経路数を小さくすることが必要である．並行経路数を小さくすることとは行列の次数を小さくすることに等しい．それは，生産性向上にはプロジェクトを構成する要素を極力少なくすることと同意である．また，開発工数の低減には，方程式で構成される図形の面積を小さくすることが必要である．したがって，生産性と同様に傾きを小さくするために 1 本平均の写像の難しさを小さくすること，そして y 切片を小さくすることが必要である．また冗長経路を低減させることも台形の面積を小さくするうえで効果が望め

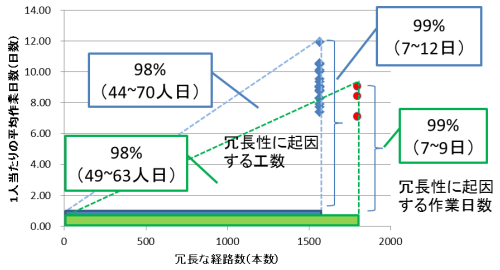



図 12 検証対象プロジェクトにおける冗長な写像経路起因の生産性と開発工数

**Fig. 12** Productivity, man-hour and redundant pathway on the game software development project.

る．なお，当然のことではあるが，開発工数は上記の面積と開発要員数の積である．したがって，そもそも開発要員数を少なくすることも開発工数を低減させるためには有用である．

### 5.3 写像経路の冗長性の低減によるプロジェクト生産性向上と開発工数低減の可能性

前節で述べた方策が実際にプロジェクトに与えるインパクトを，冗長な経路数を x 軸に，1 人あたりの平均作業日数を y 軸にとって試算し，**図 12** にその結果を示す．

すなわち，本検証によってさらなる工数低減や生産性向上を可能とする潜在的な領域の存在が確認された．前述のグラフと同様に，図中の 3 つの丸印のデータはアダプタ開発を行ったプロジェクトであり，他の 18 個の四角印のデータはそれ以外のサウンドデザイナがいないプロジェクトのデータである．どちらのプロジェクトも，図 12 に示すように点線で囲った上三角形が冗長な経路数に起因した平均作業日数や開発工数を示している．上三角形の下に位置する長方形で示された部分は，冗長性がなかった場合の平均作業日数や開発工数を示す．このように冗長性に起因した割合は非常に高い．なお，試算した結果によると，本検証プロジェクトでは写像経路 1 本あたりの平均作業工数 = 0.03〜0.05 人日/本になることが分かった．

ただし，現状このような冗長経路数に起因した工数のうち，本当に冗長な工数を判断する基準は確立できていない．たとえば，部品（Component）間の構造が，技術的な問題から図 1 の左図に示したような相互依存関係を構築することが困難な場合もある．その技術的な問題を解消することで図 1 の左図のモデルに近づける方法と，部品構造を修正することはせずに他の要素間の相互依存を低減させる方法と，たとえばこのような 2 つのアプローチが考えられる．この 2 つの選択肢のうち，どちらを選択するかによっても何を冗長な経路とするのか，その基準が異なる．したがって，その選択は前節で述べた方策をもとに現実解を個々のプロジェクトで検討し，プロジェクトにおけるその冗長性の削減に取り組む必要がある．

### 5.4 複雑性指標の有用性

4 章，および 5 章の前節までで示した実証検証の結果とその考察から，本研究で提案した複雑性指標を用いることの有用性について述べる．

4.4.2 項と 4.4.3 項で示した検証結果から，複雑性指標と生産性および開発工数が相関することを示した．さらに，5.1 節の考察で，複雑性と生産性および開発工数の因果関係を説明した．すなわち，複雑性の低減を図ることで，プロジェクトの生産性向上や，開発工数の低減が可能となることを示した．たとえば，プロジェクト開始前に，情報写像モデルを作成し，プロジェクトのメンバで，そのモデル構成について検討を行う．そして，モデルの複雑性を低減させるようなプロジェクト計画を立案する．その結果，プロジェクト完了後の実績として，複雑性を考慮しなかった場合の当初計画よりも，生産性の向上や開発工数の低減が期待できる．またさらに，派生効果として，品質の向上も見込まれる．なぜならば，5.1 節で述べたように，個々の開発者の考える時間，すなわち，設計条件を満たす解を複数案検討し，その最善案を選択するための時間を確保できるようになるからである．

現状，生産性の向上や開発工数の低減のために，たとえば，ソースコードの自動生成ツール等の方策が多くのプロジェクトで活用されている．しかし，導入したツールによって，写像経路の難しさはどれだけ低減されるのか，関連する写像経路の写像の難しさはプロジェクト全体にどれだけ効くのか，何本の写像経路に影響するのか等は十分に検討されているわけではない．4.4.4 項で述べたように，個々の要素の難しさをコントロールするだけでは，必ずしも生産性が向上するとは限らず，開発工数が低減されるとも限らない．すなわち，単に開発者が作成するソースコード量を低減させることのみに着眼した自動生成ツールの導入では，必ずしもその効果によって生産性向上や開発工数の低減が実現されるわけではない．そのようなツール導入では，逆に導入したツールの購入費やライセンス費だけ開発費が増大してしまうこととなる．一方，本研究で示した情報写像モデルによって写像経路を簡素化し，複雑性を低減させることによって，生産性や開発工数を改善することは 4 章に示したとおりである．さらに，5.3 節で述べたように，複雑性指標に着目することによって，プロジェクトには，生産性や開発工数を大きく改善する潜在的な領域が残されていることが示されている．

以上から，現在のソフトウェア開発の状況に鑑みても，本研究で示した情報写像モデルによって写像経路を簡素化し，複雑性を低減させることは，ソフトウェア開発プロジェクトの生産性を向上させ，開発工数を低減させるために有用であると考える．

## 6. 結論と今後の課題

### 6.1 結論

本研究では，ソフトウェア開発プロジェクトの要素間の複雑な関係をシンプルに表すモデルとその複雑性指標を提示し，その有用性を検証した．まず，公理的設計論をもとにしたプロジェクトモデルにより，プロジェクト内の設計情報の写像過程の指標として複雑性を定義した．その複雑性は，その写像経路の難しさと写像経路本数に依存する方程式で定義した．写像の難しさや写像経路本数は，プロジェクトモデルを構成する要素関係を表す行列により求める方法を提示した．そして，ゲームソフト開発プロジェクトをもとに上記の提案を検証した．検証の結果，提案した複雑性はプロジェクトの生産性（1 人あたりの平均作業日数）に強い相関があることが分かった．したがって，行列を用いた写像経路の難しさを算出する方法の妥当性を示すことができた．また同様に，各冗長な経路の複雑性値の総和とプロジェクトの開発工数との相関を検証し，強い相関があることを確認した．以上から，定義した複雑性や複雑性指標の総和を示す方程式の妥当性を確認した．これらの検証結果から，生産性（1 人あたりの平均作業日数）や開発工数の大半が冗長性経路により生じていることを試算し，また同時にその改善方法を示し，複雑性指標の有用性について述べた．

### 6.2 今後に向けて

本手法をもって，以下の 3 点の取り組むことにより，プロジェクトの冗長性を削減し，生産性向上・工数削減に対して大きな効果が期待できると考えられる．

① 開発者同士のコミュニケーションが極力少なく済むようにすること（冗長性を低減させるため）．

② 開発者にとって極力容易な要求条件の開発に従事させること（写像を容易にするため）．

③ 開発者数や作業数，部品数は極力少なくなるようなプロジェクト設計を行うこと（マトリックスの次数を低減させるため）．

さらにそのためには，① 各部品・機能間で極力インタフェースが少なくなるような要求仕様の整理をすることが必要である．② 開発すべき部品・機能は開発者のスキルを考慮した分担にしなければならない．言い換えれば，要求仕様と開発者のスキルを考慮した部品構成であることが必要である．③ そして，構成する部品数，開発者数が最小化する方法を選択することが必要である．これらを満たすためにプロジェクト計画時から本研究で提案した指標を活用し，より簡素化を図ったプロジェクトの体制・プロセス等を検討することが有用である．本指標の活用によって，単に設計作業が簡素化するだけでなく，その周辺作業，たとえば成果物の構成管理（版数管理）も容易になる．複雑性が低くなれば各成果物は個々の作業と関連性が薄くなり，ある作業の手戻りによって改版された設計書が他の作業に影響を及ぼすことも少なくなるからである．したがって，構成管理誤りにともなうトラブルの軽減や，各作業間の同期合わせ等の作業スケジュールの調整稼動も軽減されると考えられる．このように本複雑性指標を活用することによって，ソフトウェア開発プロジェクトの効率化の実現に貢献できれば幸いである．

また，本手法はソフトウェア開発プロジェクト以外のプロジェクトにも適用できると考えられる．今後は他分野のプロジェクトでの適用も行い，様々なプロジェクトの効率化に貢献していきたい．

**謝辞** 本研究に対し，多大なご協力をいただいたイレギュラーズアンドパートナーズ株式会社代表取締役山本一郎様，チーフディレクタ渡邊謙太様，また有益な議論をいただいた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白坂成功准教授，保井俊之特別招聘教授の各氏に感謝いたします．

### 参考文献

[1] Brooks, F.P.: *The Mythical Man-Month, Anniversary edition*, Pearson Education, MA (2010).

[2] 犬塚 篤：ソフトウェア開発における情報処理の多面性，電子情報通信学会 SWIM，ソフトウェアインタプライズモデリング，Vol.105, No.422, pp.9–14 (2005).

[3] 犬塚 篤：顧客ニーズの共有コストに関する考察，日本経営学会誌，Vol.14, pp.43–54 (2005).

[4] Sosa, M.E., Eppinger, S.D. and Rowles, C.M.: Are your engineers talking to one another when they should? *Harvard Business Review*, Vol.85, No.11, pp.133–142 (2007).

[5] von Hippel, E.: "Sticky information" and the locus of problem solving: Implications for innovation, *Management Science*, Vol.40, No.4, pp.429–439 (1994).

[6] Draft, R.L. and Lengel, R.H.: Organizational information requirements, media richness, and structural design, *Management Science*, Vol.32, No.5, pp.554–571 (1986).

[7] Eppinger, S.D.: Model-based approaches to managing concurrent engineering, *Journal of Engineering Design*, Vol.2, No.4, pp.283–290 (1991).

[8] Eppinger, S.D.: Using the design structure matrix to estimate product development time, *Proc. ASME Design Engineering Technical Conferences* (1998).

[9] Eppinger, S.D.: Innovation at the speed of information, *Harvard Business Review*, Vol.79, No.1, pp.149–158 (2001).

[10] Eppinger, S.D. and Salminen, V.: Patterns of product development interactions, *International Conference on Engineering Design, ICED 01* (Aug. 2001).

[11] Danilovic, M. and Sandkull, B.: The use of dependence structure matrix and domain mapping matrix in management uncertainty in multiple project situations, *International Journal of Project Management*, Vol.23, No.3, pp.193–203 (2005).

[12] Danilovic, M. and Browning, T.R.: Managing complex product development projects with design structure matrices and domain mapping matrices, *International*

*Journal of Project Management*, Vol.25, pp.300–314 (2007).
[13] Steward, D.V.: The design structure system: A method for managing the design of complex systems, *IEEE Trans. Engineering Management*, Vol.28, No.3, pp.71–74 (1981).
[14] 丸山勝久：日本のソフトウェア工学の今と未来，情報処理，Vol.49, No.7, pp.793–798 (2008).
[15] White, D. and Fortune, J.: Current practice in project management–An empirical study, *International Journal of Project Management*, Vol.20, No.1, pp.1–11 (2002).
[16] Sakaedani, A. and Yasui, T.: Complexity Management by Using Project Matrix, *Proc. GLOGIFT2010*, Yokohama, Japan, Paper#1569319363 (2010).
[17] Sakaedani, A. and Yasui, T.: Inter-element relations in the configured systems: Second dimension of the system complexity from the case study of the Japan's anime industry, *Proc. APCOSE 2010*, Keelung, Taiwan, Paper #272 (2010).
[18] 榮谷昭宏，狼 嘉彰，神武直彦：高品質なプロジェクトマネジメントを実現するトレーサビリティ・マトリックスの構築，シンセシオロジー，Vol.5, No.1, pp.1–16 (2012).
[19] Sakaedani, A., Yasui, T., Shirasaka, S. and Maeno, T.: A New approach to component reuse in multi-project management by using an information-centric project model, *Proc. 14th International DSM Conference*, pp.301–313 (2012).
[20] Sakaedani, A., Yasui, T. and Shirasaka, S.: Empirical validation of information centric project management: Enhancement of problem-solving capability by reduction of project complexity, *Proc. 6th International Conference on Project Management* (*ProMAC2012*), pp.751–758 (2012).
[21] Lindeman, U., Maurer, M. and Braun, T.: *Structural Complexity Management–An Approach for the Field of Product Design*, Springer, Berlin (2009).
[22] Malik, F.: *Strategie des Managements komplexer System*, Bern, Haupt (2003).
[23] Ehrlenspiel, K.: Integrierte Produktentwicklung - Methoden fur Prozessorganisation, *Produktersterllung und Konstruktion*, *3rd Ed.*, Munchen, Hanser (2007).
[24] Suh, N.P.: *Axiomatic Design: Advances and Applications*, Oxford University Press, New York (2001).
[25] Leffingwell, D. and Widrig, D.: *Managing Software Requirements A Unified Approach*, Addison-Wesley, MA (2000).
[26] Forsberg, K., Mooz, H. and Cotterman, H.: *Visualizing Project Management Models and Frameworks for Mastering Complex Systems*, *3rd Ed.*, John Wiley & Sons Inc. (2005).

**榮谷 昭宏** （学生会員）

1992 年学習院大学理学部物理学科卒業，1994 年同大学大学院自然科学研究科物理学専攻修士課程修了，同年 NTT（日本電信電話株式会社）入社．2010 年慶應義塾大学大学院システムデザイン・マネジメント研究科修士課程修了．現在，同研究科後期博士課程在籍，NTT コムウェア株式会社勤務．

**牧野 泰才** （正会員）

2002 年東京大学工学部計数工学科卒業．2004 年同大学大学院情報理工学系研究科システム情報学専攻修士課程修了．2007 年同専攻博士課程修了．同年 4 月より同専攻にて，学術振興会特別研究員．2008 年 4 月より特任研究員．2009 年 4 月より慶應大学にて特別研究助教を経て，2012 年 4 月より同特任講師．触覚情報処理の研究に従事．

**前野 隆司**

1984 年東京工業大学工学部機械工学科卒業，1986 年同大学大学院理工学研究科機械工学専攻修士課程修了，同年キャノン株式会社入社．1993 年博士（工学）学位取得（東京工業大学）．1995 年慶應義塾大学専任講師，同大学助教授を経て，現在，慶應義塾大学大学院システムデザイン・マネジメント研究科委員長/教授．