Vol.2013-SE-179 No.8
2013/3/11

既存プログラムを対象としたソフトウェア設計の理解過程の分析

本末 光[†1]　　掛下 哲郎[†1]

本研究では，既存のプログラムのソフトウェア設計の理解過程を分析する．用意したプログラムを被験者にリバースエンジニアリングさせ，そのソフトウェア設計を記述してもらった．我々はその記述過程を記録し分析を行った．これを通して，(1) ソフトウェア設計の理解に掛かる時間はソフトウェア規模よりも早く増大する，(2) 単一ルーチンプログラムではデータ構造の理解に個人差が出る，(3) オブジェクト指向プログラムでは基本設計の理解に個人差が出る，(4) 複数の設計要素を同時に理解しようとすると逆に時間がかかる，(5) 複数のクラスから構成されるプログラムの場合，クラス間関連に沿って理解することで理解プロセスを効率化できる，ことが分かった．

Analysis of Software Design Understanding Process of Existing Programs

HIKARU MOTOSUE[†1]　TETSURO KAKESHITA[†1]

We analyze software design understanding process of existing programs in this paper. Each examinee is requested to understand the given programs and to describe the design of the programs. We record and analyze the description process. Through the analysis, we found that (1) design understanding time grows faster than the program size, (2) major difference among examinees lies in the difference of understanding time of data structure in case of a single routine program, (3) major difference among examinees lies in the difference of class level understanding time in case of an object oriented program composed of multiple classes, (4) simultaneous understanding of multiple design components causes longer understanding time, and (5) understanding process of object oriented program composed of multiple classes can be optimized by understanding the classes in the order consistent with relationship among classes.

1. はじめに

近年，ソフトウェアは様々なところに利用され，大規模化が進んでいる．システムの大規模化は開発コストの増加をもたらすだけでなく，システム障害時に社会に及ぼす影響を拡大する原因になる．先に発生したみずほ銀行のシステム障害や証券取引におけるトラブルなど，一度起こると大きな損害が発生する [1)]．

ソフトウェア開発は要件定義，基本設計，詳細設計，コーディング，テスト，運用，保守といった様々な工程から成り立っている [2)]．この中で保守コストはソフトウェア開発コストの約 40%を締めている [3)]．保守コストのうち，約 60%が既存ソフトウェアを理解し，不具合の原因や修正個所を特定するために費やされる．従って，理解の容易なソフトウェアは保守コストを削減する効果があり，ソフトウェア開発全体のコスト削減にも繋がる．また，見通しのよいソフトウェア構成であれば，トラブルへの対処や復旧にかかるコストも削減できる．保守工程の割合は増加傾向にあるため [4)]，保守コストの削減は重要な課題である．

本論文では，ソフトウェア設計に焦点を当て，既存のプログラムからその設計を理解するために要する時間と理解過程を求めて分析する．これにより，設計の種類やソフトウェア設計の構造の違いによる理解に必要な時間や理解過程を実データに基づいて明らかにする．この研究を通じてソフトウェア設計の理解過程を明らかにするとともに，様々なソフトウェア工学技術の適用効果を定量的に分析することが期待される [5)]．また，個人の理解過程の分析を通じて，能力評価を行うことにより，適切な人員配置や教育・人材育成への活用が期待できる．

2. ソフトウェア設計支援ツール Perseus

Perseus は我々が開発したソフトウェア設計支援ツールである [6)]．Perseus は木構造を用いてソフトウェア設計を表現することによって系統的な設計を支援し，モジュール設計，ルーチン設計，データ構造設計，アルゴリズム構築といった様々な設計プロセスにも対応している（図 1）．

Perseus の設計木は表 1 に示す要素から構成されている．我々が実施したリバースエンジニアリング実験では，これらの要素を表に示す通りアルゴリズム，データ構造，基本設計の 3 種類に分類した．

Perseus はユーザー操作をログに記録できる．ログに記録する情報は，操作の種類，操作時刻，操作対象ノードの情報，操作の種類に応じたパラメータである．ログは利用者が設計木構造を保存すると同時に CSV ファイルに出力される．

†1 佐賀大学
Saga University

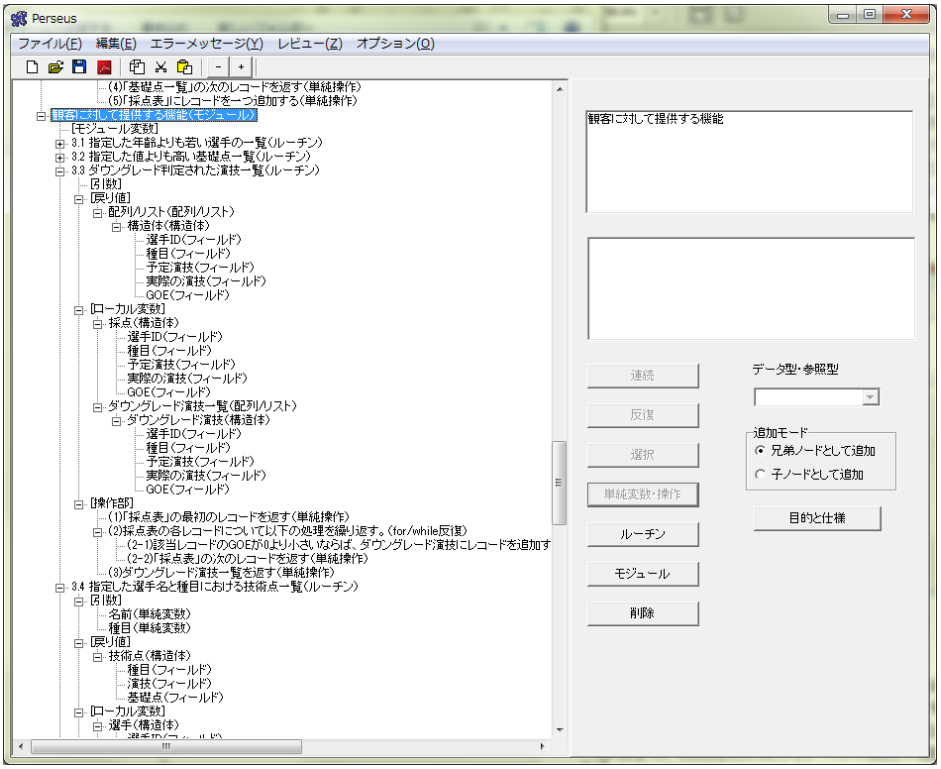

図 1　Perseus で記述したソフトウェア設計木

Figure 1　Software Design Tree Represented by Perseus

表 1　Perseus の設計要素とその分類

Table 1　Classification of Perseus Design Components

分類	Perseus の設計要素	構造化プログラムの概念	オブジェクト指向概念
基本設計	プログラム	プログラム	
	モジュール	モジュール	クラス
	ルーチン	ルーチン	メソッド
データ構造	モジュール変数	モジュール変数	インスタンス変数
	ローカル変数	ローカル変数	
	構造体, 配列/リスト, 共有体, 要素	データ構造	
アルゴリズム	if/else 選択, switch 選択, for/while 反復	制御構造	
	単純操作	文	

3. 理解過程の分析

本研究では，ソフトウェア設計の理解過程を理解所要時間と理解プロセスの 2 つの観点から分析する．本節では，理解所要時間の算出手順および理解プロセスの導出手順を説明する．

3.1 理解所要時間の算出

理解所要時間とは，既存プログラムからソフトウェア設計を理解し, 設計結果を記述するために要する時間である．理解所要時間は様々な設計要素（モジュール，ルーチン，データ構造，制御構造，基本操作など）毎に求めることができる．本研究では既存のプログラムを被験者にリバースエンジニアリングさせ，その結果を Perseus で記述する過程を記録したログを分析することで理解所要時間を求める．

Perseus が出力した設計ログにおいて，操作 A, B がこの順に記録されていた場合，操作 B に要した時間は操作 B が実行された時刻と，操作 A の時刻の差によって求める．設計木構造の各ノードに対して，当該ノードを対象とする操作に要した時間の総和を求めると，各ノードの理解所要時間を算出できる．

Perseus は設計木構造を操作するために様々な機能を提供しているが，それらの機能は以下に示す 3 種類に分類される．理解所要時間の算出は機能ごとに多少異なる．

木構造の編集機能：編集機能には，モジュール，ルーチン，データ構造，制御構造等に対応する部分木の追加，個別ノードの編集（設計テキストの変更，データ型・参照型指定の変更）がある．これらの操作に対しては，操作が記録された時刻と，その直前の操作の時刻の差を理解所要時間とする．

木構造の複製機能：複製機能には，設計部分木のコピー・カット（切り取り）・ペースト（貼り付け）の各操作がある．コピー（カット）& ペーストの理解所要時間は，コピー（カット）に要する時間と貼り付けに要する時間の和により求める．これはコピー操作とペースト操作の間に別

の操作が挿入される場合も考慮したためである．コピーに要する時間は，コピー操作とその直前の操作の時刻の差とする．カット操作やペースト操作に要する時間も同様に，直前の操作の時刻との差によって算出する．

ノード等を対象とする参照操作：参照操作には，選択ノードの変更，ノードまたは設計部分木の縮約・展開が含まれる．これらの操作に要する時間は，対象となるノード等を理解するために要する時間として理解所要時間に含める．

3.2 理解プロセスの導出

理解プロセスは，設計要素毎に理解開始時刻と理解終了時刻を求めてガントチャートで表現したものである．これを用いて理解過程の特徴を分析する．

ガントチャートを作成するためには，設計要素毎に理解開始時刻と理解終了時刻を求める必要がある．

設計要素 x の理解開始時刻は，x に対応する「部分木の追加」をログ中で検索し，その直前に実行された「選択ノードの変更」操作の実行時刻によって定義する．これは，設計要素の挿入位置や種類を理解するための時間を考慮するためである．

一方，設計要素 x の理解終了時刻は，x に対する「設計テキストの変更」ないしは「データ型/参照型の変更」操作のうち，ログ中で最後に出現したものの実行時刻によって定義する．

上記は設計木の葉ノードを対象とした定義であるが，データ構造やルーチン等のように部分木に対応する設計要素の理解開始時刻と理解終了時刻は以下のように定義する．設計部分木の理解開始時刻は，その部分木を構成する葉ノードの理解開始時刻のうちで最も早いものとする．一方，設計部分木の理解終了時刻は，その部分木を構成する葉ノードの理解終了時刻のうちで最も遅いものとする．

我々は Perseus のログ作成機能にログ分析機能を追加した．ログ分析機能は，ログから設計木構造を再現する機能と分析した理解プロセスを出力する機能からなる．

ログから設計木構造を再現する際には，全ノードに対して理解開始時刻と理解終了時刻を計算する．この際に，リバースエンジニアリングで用いたプログラム毎に，設計木構造の正解例に基づいてノードの種類を正規化することで，被験者毎の設計木の細かな違いを吸収し，相互に比較できるように工夫している．

また，分析した理解プロセスを出力する機能では，設計木のノード毎にノード名，階層レベル，ノードの種類，理解開始時刻，理解終了時刻と理解開始時刻の差，理解終了時刻と部分木の理解終了時刻の差を出力する．Microsoft Excelでこのデータを処理することでガントチャートを作成する [7].

4. リバースエンジニアリング実験

既存のプログラムにおける設計の理解過程を分析するためにリバースエンジニアリング実験を行った．本実験では，佐賀大学知能情報システム学科の 3 年次専門科目「システム開発実験」の履修者 7 名に 3 種類のプログラムを与えてリバースエンジニアリングを行わせた．被験者に与えた 3 種類のプログラムの概要を表 2 に示す．被験者の学生は 1 年次後期から 2 年次前期にかけて C++を用いた構造化プログラミングを，2 年次後期から Java を用いたオブジェクト指向プログラミングを学んでおり，学生にとってもなじみがある．また，2 年次後期には Perseus を用いたソフトウェア設計演習も行っている．

表 2　実験で用いたプログラムの概要
Table 2　Summary of the Programs used at the Experiment

プログラム	行数	モジュール（ルーチン）数	プログラミング言語	備考
Program A	29	1（1）	C++	構造化プログラム
Program B	27	1（1）		
Program C	72	6（14）	Java	OOP

被験者は Perseus を用いて設計木構造を作成する．各被験者が与えられたプログラムを正しく理解していることは，Perseus で作成した設計木構造によって確認している．リバースエンジニアリングの過程は Perseus のログ機能を用いて記録する．このログを 3.1 節で述べた方法を用いて分析することで，各被験者の理解所要時間をノードの種類毎に算出した．また，3.2 節で述べた方法により，ガントチャートを用いて理解プロセスを可視化した．

5. 理解所要時間の分析

5.1 集計データの分析

表 3 は 3.1 節で説明したリバースエンジニアリング実験の結果である．7 名の被験者からデータを集め，総理解所要時間及び設計の種類毎に総理解所要時間に締める比率（理解所要時間比率）を示す．

Program A 及び Program B の理解過程において基本設計に関する理解所要時間の割合は低い．これは，メイン関数のみのプログラムであるため，理解しやすいことが原因と考えられる．

また, Program C の総理解所要時間の平均値は Program A の 3.7 倍，Program B の約 4 倍だが，表 2 を見ると行数はそれぞれ 2.48 倍と 2.66 倍に留まっている. このことから，理解所要時間はソフトウェア規模よりも早く増大することが推定される．これはソフトウェア設計の理解には個々の要素だけではなく，要素間の関連まで理解する必要があるためである．このため，理解所要時間の増大はこの関連の

表 3　理解所要時間の集計結果
Table 3　Summary of Understanding Time

			Program		
			A	B	C
総理解所要時間（秒）		平均	730	679	2714
		標準偏差	442	140	577
理解所要時間比率（%）	アルゴリズム	平均	63.0	54.1	30.8
		標準偏差	19.3	13.9	7.6
	データ構造	平均	33.8	44.4	29.1
		標準偏差	16.7	13.8	7.2
	基本設計	平均	3.1	1.4	39.9
		標準偏差	3.7	1.1	10.4

増大に沿ったものだと考えられる．

Program A は他の 2 つと比較して単位行数あたりの総理解所要時間のばらつきが大きい．被験者は Program C の設計を理解する過程で基本設計の理解に最も多くの時間を使っており，アルゴリズムとデータ構造における理解所要時間の標準偏差は比較的小さい．

能力の低い被験者の場合，与えられたプログラムを理解するためにはより多くの時間が必要になる．また，プログラム理解を丁寧に行えばより多くの時間がかかる．そのため，理解所要時間のばらつきは，被験者の能力差や理解の丁寧さなどに起因していると考えられる．

Program C は他の 2 つより空白行が少ない．通常，空白行は設計の区切りに用いられるため，Program C にどのような影響があるのか分析する必要がある．また，Program C のみがオブジェクト指向プログラムであることから，それによる影響も考慮しつつ分析する．

5.2 アルゴリズムに関する理解所要時間の分析

図 2 では，総理解所要時間とアルゴリズムに関する理解所要時間の関係をプログラム毎に示す．

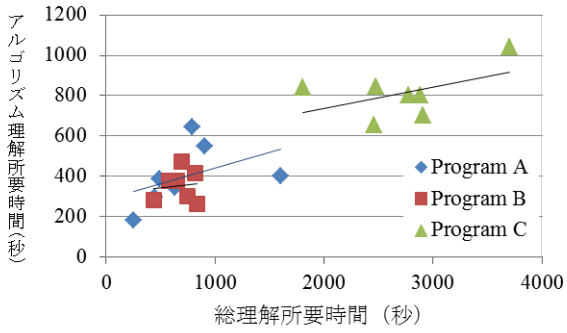

図 2　総理解所要時間とアルゴリズムに関する理解所要時間の関係
Figure 2　Correlation between Total Understanding Time and Algorithm Understanding Time

総理解所要時間とアルゴリズムに関する理解所要時間の間の相関係数は，Program A では 0.44，Program B では 0.11，Program C では 0.48 であり，両者の相関は強いとは言えない．これはアルゴリズムのリバースエンジニアリングが，与えられたプログラムをトレースするだけの比較的単純な作業であり，それだけでは与えられたプログラムを十分には理解したとは言えないためだと考えられる．

ただし，Program A において時間をかけて丁寧な理解を行った被験者が 1 名存在する．当該被験者を除外して計算した場合，相関係数は 0.90 となることから，Program A における理解所要時間のばらつきは，アルゴリズムに関する理解所要時間の違いによって説明可能だと考えられる．

なお，Program C におけるアルゴリズム理解所要時間は 812 秒であり，Program A の 401 秒，Program B の 355 秒と比較して，単位プログラム行数当たりのアルゴリズム理解に要する時間はむしろ少ない．これは，Program C が複数のクラスおよびメソッドに分割されており，個別のメソッドのアルゴリズムを理解する作業が容易なことを反映していると考えられる．

5.3 データ構造に関する理解所要時間の分析

図 3 では，総理解所要時間とデータ構造に関する理解所要時間の関係をプログラム毎に示す．Program A および B においては両者の間に正の相関が見られる．総理解所要時間とデータ構造に関する理解所要時間の間の相関係数を計算すると，Program A では 0.95，Program B では 0.88，Program C では 0.53 である．これは，単一ルーチンのプログラムにおいて，使われている変数間の関連や，保持するデータ値の変遷を理解するプロセスは，アルゴリズムをトレースするプロセスよりも難易度が高く，被験者の能力差が出やすいためと考えられる．

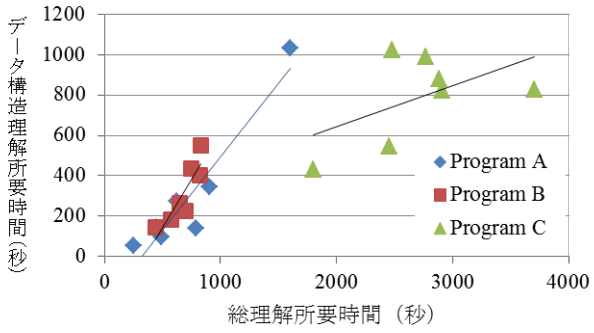

図 3　総理解所要時間とデータ構造に関する理解所要時間の関係
Figure 3　Correlation between Total Understanding Time and Data Structure Understanding Time

Program A において，5.2 節で言及した特殊な被験者を除

外すると，データ構造の理解に要する時間の平均値は 175.5 秒である．これに対して，Program B におけるデータ構造に関する理解所要時間の平均値は 314.8 秒であり，両者には有意な差がある．Program A とは異なり，Program B はユーザー定義のデータ型として構造体を定義して使用しており，この点がデータ構造に関する理解所要時間が長くなる原因の一つと考えられる．

5.4 基本設計に関する理解所要時間の分析

図 4 では，総理解所要時間と基本設計に関する理解所要時間の関係をプログラム毎に示す．

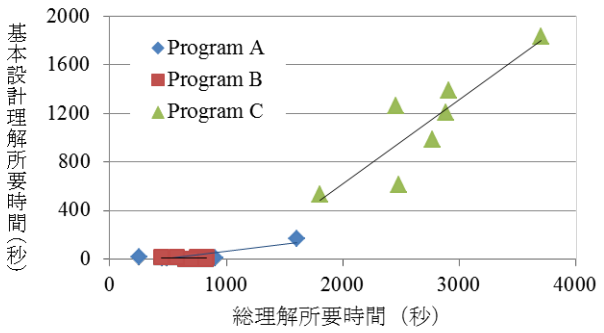

図 4　総理解所要時間と基本設計に関する理解所要時間の関係
Figure 4　Correlation between Total Understanding Time and Basic Design Understanding Time

総理解所要時間と基本設計に関する理解所要時間の間の相関係数は，Program A では-0.57，Program B では-0.14，Program C では 0.88 であり，Program C において特に強い．これは Program C が 6 個のクラスおよび 14 個のメソッドから構成されているためであり，これらの要素間の関連を理解しなければ，プログラムの設計を理解できないためと考えられる．これに対して，Program A および B は単一の関数で構成されているため，図 4 からも明らかなように，基本設計の理解にはほとんど時間を要しない．

また，Program C はオブジェクト指向プログラムであり，プログラムが対象とするデータ構造とクラス構造やクラス間の関連が対応している．このため，基本設計を理解することがデータ構造を理解することにつながる．Program C において，データ構造に関する理解所要時間と総理解所要時間の間の相関係数（0.53）があまり高くないのは，基本設計の理解とデータ構造の理解に共通部分があるためだと考えられる．

6. 理解プロセスの分析

6.1 単一ルーチンプログラムの理解過程分析

本節では，理解所要時間の長短に応じて理解過程がどのような特徴を持つのかを，単一ルーチンプログラムである Program A 及び Program B のガントチャートに基づいて考察する．

今回 7 人分の理解過程を分析したが，理解所要時間が長いもののいくつかは，ローカル変数の理解が不十分なままアルゴリズムへの理解に移っていた．Program A では被験者 B，Program B においては被験者 B と被験者 E が該当する．また，被験者 C は，理解結果を詳細に記述したため理解結果の作成に時間がかかっていた．

ガントチャートを観察すると，3 つの種類に分類できる．以下のその種類と特徴を述べる．但し，分類の際には手戻りにより発生した操作は除いて考える．

深さ優先型：与えられたプログラムを上から順を追って理解し，設計を記述する手順（図 5）．Perseus で収集したログは「選択ノードの変更（ノードを追加する位置を決める）」⇒「ノードの追加」⇒「設計テキストの変更」の繰り返しからなっている．ガントチャートは理解期間の重なりがないことから非常に分かりやすく，各設計要素の理解所要時間も容易に可視化される．

幅優先型：兄弟関係にある設計要素について一括して先にノードを作成し，その後，設計テキストを書く手順（図 6）．ログでは，複数の「部分木の追加」が行われた後で，設計テキストを書き換えるノードに「選択ノードを変更」し，追加していった部分木とノードに合わせて設計テキストの書き換えを記述していく．この時，木構造の幅優先のように，同階層のノードを全て作成するのではなく，参照しているノードの入れ子のみを作成していく．このため，部分的に木構造が出来上がる．ガントチャート上では複数の理解期間が重なるため，手戻りと訂正の区別が付きにくいこともあるが，それぞれある一定時間で設計要素が 1 つずつ理解終了時刻を迎える．この時の一定時間は 1 つのノードを設計する時間と同程度である．

混合型：データ構造とアルゴリズムの理解を異なる手順で理解する（図 7）．例えば，データ構造の理解では幅優先，個別ルーチンのアルゴリズム理解は深さ優先といったケースがある．理解手順は被験者によって異なるため，個別に特徴を調べる必要がある．

上述した理解の手順を理解ストラテジーと呼ぶ．プログラムと理解ストラテジー毎に，被験者の理解プロセスを分類した結果を表 4 に示す．被験者 D は 2 つのプログラムとも深さ優先型で，被験者 B と F は幅優先型で理解している．しかし，その他の被験者は，プログラムによって理解スト

ラテジーを使い分けている．

表 4　理解ストラテジーによる理解プロセスの分類
Table 4　Classification of the Understanding Process based on Understanding Strategy

プログラム	深さ優先型	幅優先型	混合型
Program A	D，E，G	B，C，F	A
Program B	A，D	B，F	C，E，G

表 4 に示す 6 通りの事例から典型的なガントチャートを 3 種類選んで図 5〜7 に示す．ガントチャートにおいて，赤のチャートバーは当該ノードに対応する設計要素の理解期間を示す．一方，緑のチャートバーは，基本設計の各設計要素，データ構造，アルゴリズム等の設計部分木に対する理解期間を示す．そのため，設計部分木の要素となるノードの理解期間を全て含む．また，ガントチャートの縦軸は設計要素に対応するが，併記している数値は Perseus の設計木における当該要素の階層レベルを表す．横軸の単位は秒である．

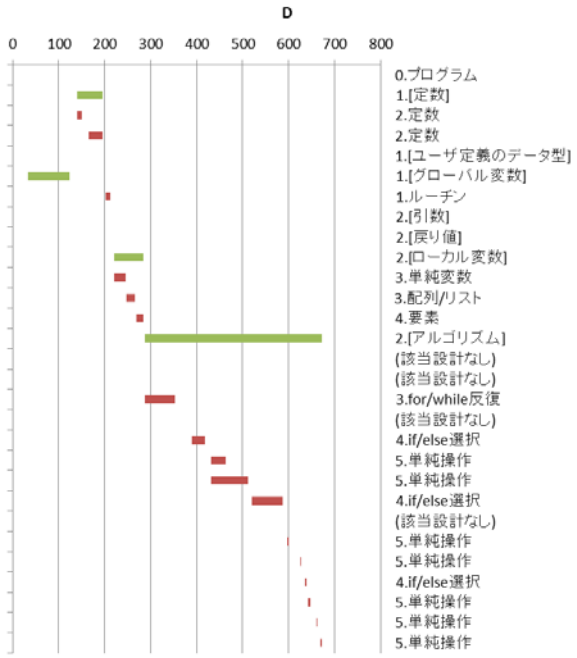

図 5　Program A に対する被験者 D の理解プロセス
Figure 5　Understanding Process of Examinee D for Program A

表 5　理解ストラテジーによる平均理解所要時間の変化
Table 5　Relationship between Average Understanding Time and Understanding Strategy

プログラム	深さ優先型	幅優先型	混合型
Program A	554.7 秒	985.0 秒	493.0 秒
Program B	506.5 秒	733.5 秒	759.0 秒

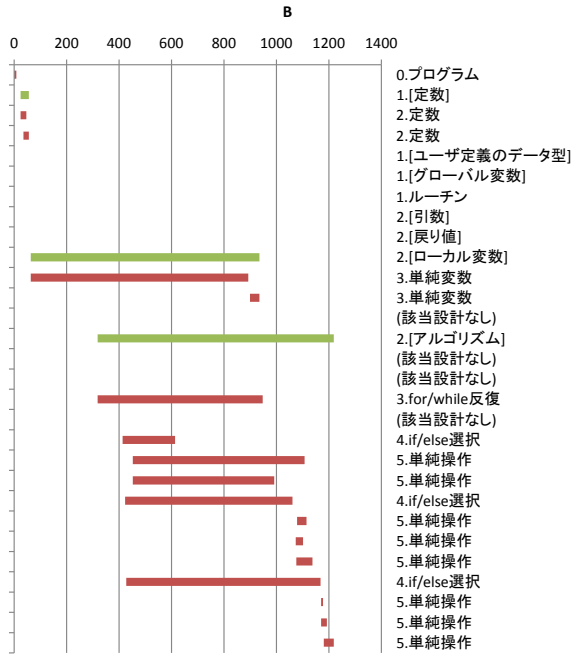

図 6　Program A に対する被験者 B の理解プロセス
Figure 6　Understanding Process of Examinee B for Program A

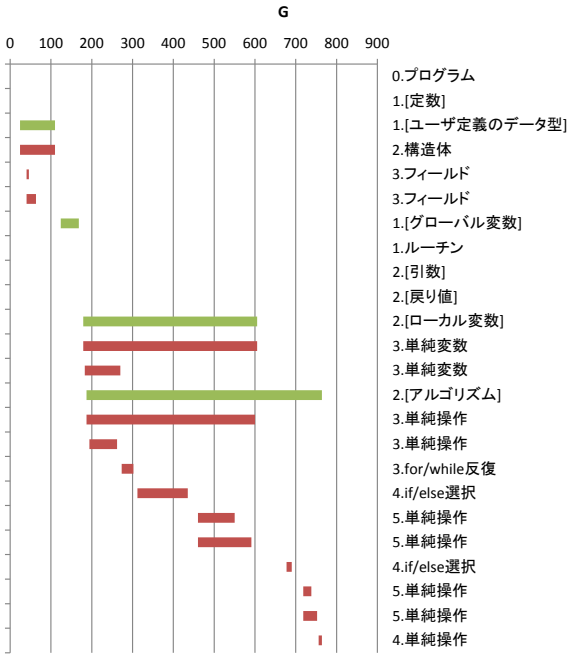

図 7　Program B に対する被験者 G の理解プロセス
Figure 7　Understanding Process of Examinee G for Program B

表 5 にはプログラムと理解ストラテジーごとに，理解所要時間の平均値を示す．Program A における理解所要時間の平均は混合型，深さ優先型，幅優先型の順となった．また，Program B では深さ優先型，幅優先型，混合型の順となった．Program A において，深さ優先型は混合型より一

分近く長いが，Program B では順位が逆転して約 250 秒の差がついている．また，幅優先型においては Program A よりも Program B の方が良い結果が得られた．

以上のことから，深さ優先型は安定して短い理解所要時間でソフトウェア設計を理解できると考えられる．一方，単純なプログラムの場合，幅優先型はより短い時間で理解できるが，プログラムが複雑化すると理解所要時間が長くなると考えられる．混合型は，理解所要時間が長くかかってしまう場合がある．

このことから，理解ストラテジーの中では深さ優先型が最も効率が良いと考えられる．混合型は場当たり的に自分が理解した順番を記述していると予想したが，Program A においては幅優先型の方がより長い時間を要している．

このような結果になる理由としては，深さ優先型の理解プロセスでは被験者が狭い範囲に注意を集中するのに対して，幅優先型や混合型の理解プロセスでは，被験者が複数の設計要素を並行して理解しようとするためと考えられる．

表 6　理解ストラテジーによる平均訂正数の変化
Table 6　Relationship between Average Number of Correction and Understanding Strategy

プログラム	深さ優先	幅優先	混合型
Program A	0.67	2.33	1.00
Program B	1.00	2.00	2.33

表 6 は理解プロセス当たりの訂正数を理解ストラテジーごとに集計したものである．訂正回数は，修正された設計要素数をカウントする．したがって，同じ設計要素に複数回の訂正が発生しても 1 回とカウントする．このため，訂正は対象の設計要素よりも後の設計要素の理解が完了した後に理解期間が存在する場合をカウントする．

表 5 と表 6 を比較することにより，平均訂正数が多い場合，平均理解所要時間もより長くなることが分かる．これは，手戻りが増えることが主な原因である．

6.2 複数クラスの理解過程分析

本節では，理解所要時間と理解過程がどのように関係しているのかを，オブジェクト指向プログラムのガントチャートと理解所要時間に基づいて分析する．

Program C は 2 つの継承関連と 2 つの集約関連を含む．これらの関連とクラス間の理解順序が理解所要時間にどのような影響を与えたかを分析する．

表 7 はクラス間の継承関連に沿った順序に従ってクラスの理解を始めた場合と，そうでない場合の比較結果を示す．この表から，継承関連に沿った順序でクラスの理解を始めた場合，総理解所要時間がより短縮されることが分かる．

表 7　クラス理解の開始順序と継承関連
Table 7　The Order of Initiating Class Understanding and Inheritance Relationship among Classes

	継承関連に沿った順序	継承関連に沿わない順序
被験者	A，B，D，F，G	C，E
平均理解所要時間	2642.2 秒	2896.0 秒

表 8 にはクラス間の集約関連に沿った順序に従ってクラスの理解を終えた場合と，そうでない場合の比較結果を示す．クラス間の集約関連に沿った順序でクラスの理解を終えた場合，そうでない場合よりも平均 571.2 秒も理解所要時間が短縮される．クラス間の集約関連に沿った順序に従ってクラスの理解を開始した場合と，そうでない場合の比較も行ったが，平均理解所要時間の差は 40 秒程度だった．これは，集約関連に沿わない順序でクラス理解を開始したとしても，最終的に集約関連に沿った順序で理解すれば，理解所要時間が長くなることはないことを示している．

表 8　クラス理解の終了順序と集約関連
Table 8　The Order of Ending Class Understanding and Aggregation Relationship among Classes

	集約関連に沿った順序	集約関連に沿わない順序
被験者	C，D，G	A，B，E，F
平均理解所要時間	2388.3 秒	2959.5 秒

継承関連・集約関連の両者に沿った順序でクラスを理解した場合のガントチャートを図 8 に示す．一方，どちらの関連にも沿わない順序で理解を進めた場合のガントチャートを図 9 に示す．これらの図からも，本節での議論の正しさを確認できる．

これらのことから，クラス間の関連に沿った順序でクラスを理解するのが，理解プロセスを効率化する上で合理的なことが分かる．

7. おわりに

我々はソフトウェアを理解するために要するコストを定量的に計量するための研究を行っている [8,9]．先の研究において，ソフトウェアの理解容易性を計量するために理解コストの概念を提案した．本論文では，ソフトウェア設計に焦点を当て，既存のプログラムからその設計を理解するために要する時間と過程を求めて分析した．これにより，設計の種類やソフトウェア設計の構造の違いによる理解に必要な時間や理解過程を実データに基づいて明らかにした．

本研究では，(1) ソフトウェア設計の理解に掛かる時間はソフトウェア規模よりも早く増大する，(2) 単一ルーチンプログラムではデータ構造の理解に個人差が出る，(3)

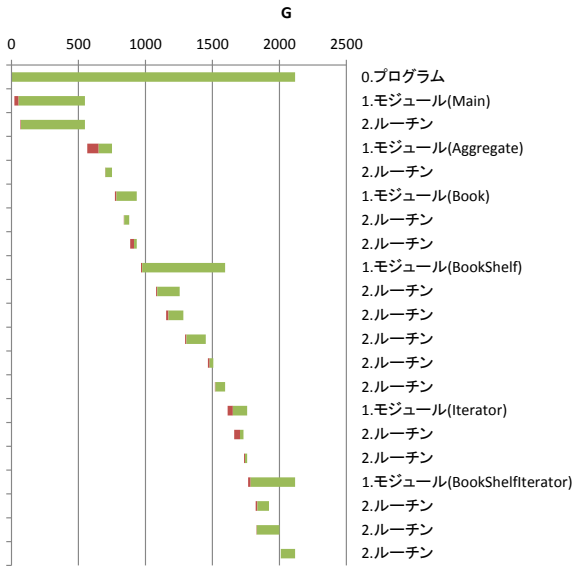

図 5　Program C に対する被験者 G の理解プロセス
Figure 5　Understanding Process of Examinee G for Program C

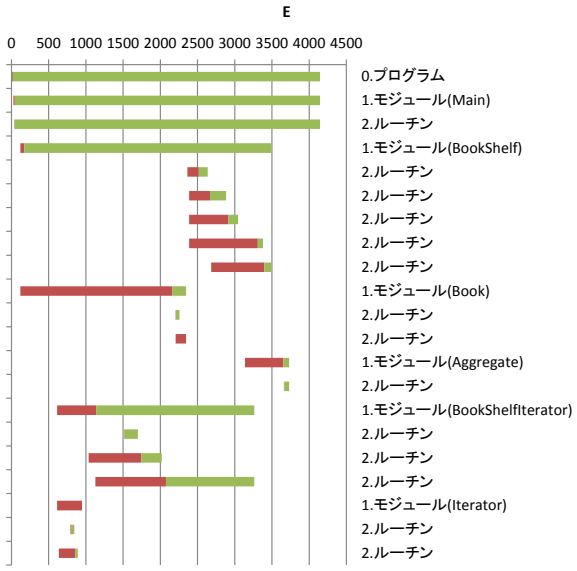

図 6　Program C に対する被験者 E の理解プロセス
Figure 6　Understanding Process of Examinee E for Program C

オブジェクト指向プログラムでは基本設計の理解に個人差が出る，(4) 複数の設計要素を同時に理解しようとすると逆に時間がかかる，(5) 複数のクラスから構成されるプログラムの場合，クラス間関連に沿って理解することで理解プロセスを効率化できる，ことが分かった．

今後の課題としては，より多くの被験者から理解過程データを収集して分析を行うことが挙げられる．また，理解所要時間の長短に基づくソフトウェア設計の定量的な評価や，様々なソフトウェア工学技術の評価に役立てるための研究が挙げられる．さらに，個人の理解プロセスを分析することにより，個人の能力評価を行い，適切な役割分担に役立てるとともに，教育や人材育成の際に活用することもできる．

参考文献

1) 玉井哲雄, ”ソフトウェア社会のゆくえ”, 岩波書店, 2012.
2) NTT データソフトウェア工学推進センタ, ”実例で学ぶソフトウェア開発”, オーム社, 2008.
3) C. McClure, The Three Rs of Software Automation: Re-engineering, Repository, Reusability, Prentice-Hall, 1992. （邦訳：ベスト CASE 研究グループ, “ソフトウェア開発と保守の戦略”, 共立出版, 1993）
4) 山本久志, “情報システムの保守工程におけるデータ分析”, 経済科学研究所 紀要, 第 41 号, 39－49, 2011.
5) 本末光, 掛下哲郎, “ソフトウェア設計支援ツール Perseus を用いたソフトウェア設計の理解過程の分析”, 情報処理学会九州支部 火の国情報シンポジウム, 2012.
6) T. Kakeshita, T. Fujisaki, “Perseus: An educational support tool for systematic software design and algorithm construction”, Proc. CSEE&T 2006.
7) Microsoft Corporation, “Excel でデータをガントチャートで表示する- Excel - Office.com”, http://office.microsoft.com/ja-jp/excel-help/HA010238253.aspx, 1999.
8) 山崎直子, 掛下哲郎, ”認知心理学的アプローチに基づくソフトウェア理解度計量法”, コンピュータソフトウェア, 16(6), 571-583, 1999.
9) 山崎直子, 掛下哲郎, ”静的なオブジェクト指向プログラムに対するインスタンスを考慮した理解コストの計量法”, コンピュータソフトウェア, 20(4), 331-344, 2003.