プログラム理解のための付加注釈 DocumentTag の提案

田中 昌弘† 石尾 隆† 井上 克郎†

† 大阪大学 大学院情報科学研究科
〒 560-8531 大阪府豊中市待兼山町 1-3

オブジェクト指向プログラミングでは，カプセル化や継承・多相性などの機構によって，情報隠ぺいを実現している．その結果，ある 1 つのプログラム要素の詳細な動作を理解したいという場合であっても，継承やメソッド呼び出しによって関連するモジュールを調査する必要がある．これに対して，プログラム理解支援を目的としたツールや手法は多数提案されているが，開発者が理解した内容を効果的に記録・再利用する手段はあまり提供されていない．そこで本研究では，ソースコードの読解において理解した内容を効果的に記録するための付加注釈 DocumentTag を提案する．DocumentTag は，ソースコード上の識別子に対して直接記述する注釈であり，また，他の関連する識別子へと同一の注釈を関連付ける注釈伝播ルールを備えたものである．DocumentTag を統合開発環境 Eclipse の拡張機能として実装し，注釈伝播が情報の提示に有効であることを確認した．

DocumentTag: Source Code Documentation Environment to Support Program Comprehension

Masahiro Tanaka† Takashi Ishio† Katsuro Inoue†

† Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

Object-Oriented Programming provides several mechanisms such as encapsulation, inheritance and polymorphism for developers to hide implementation detail of a functional unit. Developers who want to inspect precise behavior of a module have to inspect a number of modules that related to the module using class hierarchy and method call relation. While many approaches are proposed to support program comprehension, a few tools focus on documentation of the result of comprehension tasks. In this paper, we propose a new documentation named DocumentTag for developers to write a document for an identifier. DocumentTag has propagation rules that propagates the document of an identifier to other related identifiers. We have implemented DocumentTag as an Eclipse plug-in and conducted a case study. The result shows that DocumentTag effectively provides documents to developers.

1 まえがき

近年普及しつつあるオブジェクト指向プログラミングでは，カプセル化や継承・多相性などの機構を効果的に用いることで，コーディング，テスト，再利用の効率を向上させることができる [3].

一方で，オブジェクト指向プログラミングを導入することにより，クラス間の継承関係や，動的束縛といった新たなプログラム内の依存関係を把握しながらプログラム理解を行うことが必要となる．この依存関係を把握するには，ツールによる理解支援が不可欠であるといわれている [8][12].

プログラム理解支援を目的としたツールや手法は多数提案されているが [1][5][7]，理解した内容を記録・再利用する手段が効率的でないことが課題として残されている [6][9][10]．たとえば，コメントとしてプログラム中に多数の注釈を散在させると，プログラムの可読性が低下し，バグを発生させる危険がある [4]．また，プログラムとは独立した文書として記述する場合，ソースコードの変更に追従できずにプログラムとの不整合が生じることがある．

本研究では，開発者がソースコードの読解において理解した内容を効率的に記録するための付加注釈 DocumentTag を提案する．DocumentTag とはプログラムの識別子と関連付けられた注釈であり，開発者は識別子情報を入力として，注釈を参照することができる．また，DocumentTag は他の関連する識別子へと同一の注釈を関連付ける注釈伝播ルールを備えている．開発者がプログラムに含まれる識別子に対して，仕様や使用例などの注釈を DocumentTag として記録すると，その識別子だけでなく他の関連する識別子に対しても，記録した注釈が「関連する注釈」として付加される．たとえば，あるクラスに対して，開発者がクラスの役割を DocumentTag として記述すると，そのクラスだけでなくサブクラスに対しても関連付けが行われ，開発者がサブクラスの注釈を参照する際に，関連付けされた DocumentTag も同時に参照することができる．これにより開発者は，識別子間の関連を把握しながら効率的なプログラムの理解を行うことができる．

DocumentTag の有効性を確認するために，DocumentTag の記述・閲覧ができる機能を持った Eclipse プラグインの実装を行い，DocumentTag による注釈伝播の有効性を評価する適用実験を行った．具体的には学生 4 人に対して，Java 言語で書かれた既存のプログラムの拡張作業を実施し，注釈伝播が識別子間の関連を把握する上で効果的であったことを確認した．

2 背景

オブジェクト指向プログラミングを導入することで，プログラム理解が困難になるという問題の主要な原因は，クラス階層が複雑になり，開発者がソースコード上を移動する負担が増えることにある．この問題に対処するために，開発者が与えた情報を基にソースコードの移動を支援する研究がいくつか行われている．

Kersten らは，開発者が閲覧または編集を行っているメソッドやクラスに対して Degree Of Interest というメトリクスを計算し，頻繁に操作しているメソッドやクラスの集合だけを開発者に提示する手法を提案している [7].

Storey らは，開発者がソースコードコメント中に検索用のタグを埋め込む TagSEA というツールを提案している [10]．コメントに’@tag’ で始まる文字列を開発者が記述することで，記述した内容が，位置情報とともに，注釈一覧として表示される．一覧から登録した位置に移動できるので，検索を行わずに効率的にソースコード内を移動できる．しかし適用実験の結果としては，開発者がコメントの中に情報を直接記述することや，記述した内容をグループ間で共有することに抵抗があることが判明している．その結果，記述した内容が，効率的に再利用されないままとなっている．

Oezbek らはソースコード上の移動を支援する Eclipse の拡張機能として JTourBus というツールを提案している [9]．JTourBus は，開発者がコメント中に’@JTourBusStop’ で始まる文字列を記述することで，ソースコードを巡回する「ツアー」の 1 要素として登録することができ，開発者がソースツアーを選択すると，登録した’@JTourBusStop’ の箇所に順を追って自動的に移動する．ソースコード上の移動を自動化することで移動のための検索をする作業が軽減され，プログラム理解を効率的に

行うことができる．適用実験の結果として，コメントよりも'@JTourBusStop' として記述する方が，保守作業が効率化されるという結果にはなったが，より多くのメタ情報を記述できる Javadoc の方を利用する方が効率的であるという結果が得られた．その原因としては，JTourBus で記述した内容に対して，有効な再利用方法が確立できず，'@JTourBusStop' として記述した内容が効率的に再利用できなかったことが挙げられている．

以上より，プログラム理解を目的とした手法やツールは提案されているが，プログラムに直接記述することへの負担がある点や，記述した内容を開発者間で共有・再利用することが困難である点が課題として挙げられている．また，識別子間の依存関係を基にした移動支援は行われず，ツールを用いたプログラム理解支援を行うには課題が残されている．本研究では，この課題に対して，識別子間の関連を把握しながら，開発者の記述内容を効果的に共有・再利用する手段として，DocumentTag を提案し，Eclipse プラグインとして実装した．

3 付加注釈 DocumentTag の提案

DocumentTag は，プログラムの識別子に関連付けて記述する注釈であり，開発者は識別子情報を入力として，注釈を記録・参照する．本研究での注釈とは，プログラムに含まれる識別子に対して，仕様や使用例などを記述した自然言語の文書である．また，DocumentTag は注釈伝播ルールを備えている．注釈伝播とは，注釈を DocumentTag を用いて記録することで，その識別子だけでなく関連する識別子に対しても，関連する注釈として付加されることである．これにより，開発者は直接関連付けられた識別子だけでなく，依存関係を持つ識別子に対しても注釈を参照することができる．

3.1 付加注釈 DocumentTag の定義

DocumentTag は，以下のデータ項目から構成される．

- 関連付けられている識別子 ident
- 自然言語からなる注釈 text
- 注釈のカテゴリ label
- 注釈の言語（ロケール） lang
- 記録した日時 time
- 著者名 author
- 注釈伝播ルールの種類 type

識別子 ident は，プログラム中に出現する任意の識別子であり，変数，メソッド，クラスを意味する識別子が該当する．ここで，識別子は，宣言によって区別される．たとえばクラスを意味する識別子 `ArrayList` はソースコードの複数の場所に出現しうるが，開発者がどの識別子の出現を選んだとしても，プログラムが参照するただ 1 つの `ArrayList` クラスの宣言に対して注釈が付与される．これによって，DocumentTag は，同一の宣言を参照するすべての識別子の出現で，ソースコード上の定義位置に関係なく，情報を参照することができる．この点は，Javadoc などの支援ツールに類似しているが，ローカル変数を含む任意の識別子に付与可能であるという点が異なる．

DocumentTag はソースコードに直接記載されることはなく，宣言に関連付けられた別データとして管理される．そのため，複数の開発者が同一の識別子に独立した注釈を付与することができる．それらの注釈を区別，管理するために，記録した日時，著者名も DocumentTag 情報の一部として管理する．

実際に text として記述する内容は，コメントや仕様書などに記述する内容と同様であり，具体的には以下のようなものを想定している．

- 識別子の仕様
  - プログラム内での役割や，一般的な扱い方，注意点，他の識別子との関連を記す
- 識別子の使用例
  - 具体的な使い方をコード例を用いて説明する
- バグレポート
  - バグ ID やバグの内容，再現手順を記述する

これらの種別を，開発者が検索，管理できるよう，注釈のカテゴリ label を付与できるようにしている．

また，注釈伝播ルールの種類 type は，後述の注釈伝播を用いて開発者に関連する注釈を提示する際に用いられる情報である．たとえば，親クラスから注釈が伝播された場合に，親クラスから注釈が伝播されたことを示す'Super' が type の値となる．

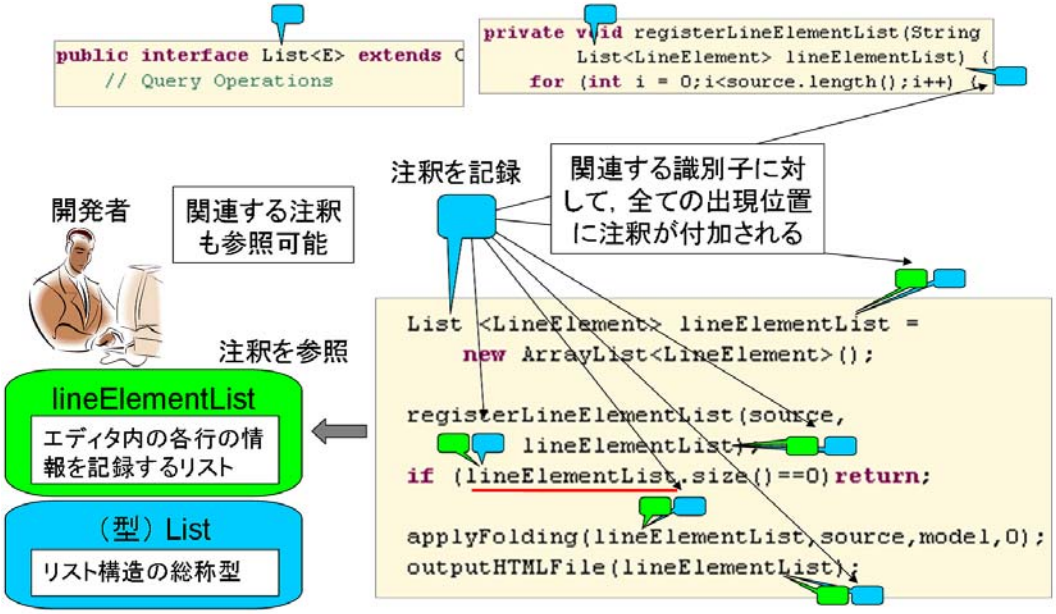

図 1: 注釈伝播

3.2 注釈伝播

オブジェクト指向プログラムを理解する際，重要となるのが，クラス間の継承関係や動的束縛といった，識別子間の依存関係である．たとえば，あるメソッドの意味を調べるには，そのメソッドの引数の型や戻り値の型の仕様を調べ，また，そこにクラスが指定されていれば，そのクラス定義や継承関係を調べるといった作業が生じる．

本研究では，識別子に対して付加された DocumentTag を，関連する識別子へと自動的に関連付ける注釈伝播 (Document Propagation) を提案する．ある識別子を開発者が参照するとき，その識別子に直接付与されている DocumentTag だけでなく，関連する識別子に付与されている DocumentTag も同時に開発者へと提供することで，効果的なプログラム理解を行う．このことをコード例を用いて説明する．

図 1 は注釈伝播の模式図を表現したものである．開発者が List クラスに注釈を記録すると，プログラム内に出現する全ての List という識別子に注釈が付加される．加えて，List クラスと関連のある識別子にも注釈が付加される．図 1 では，局所変数 lineElementList は，List 型で宣言されているため，List クラスと関連があると判断され，プログラム内の局所変数 lineElementList のすべての出現にも注釈が付加される．開発者が局所変数 lineElementList の注釈を参照すると，局所変数 lineElementList と関連のある List クラスの注釈も同時に参照することができ，プログラム内の依存関係に沿った効率的な理解を行うことができる．注釈伝播は，記録した 1 つの注釈がプログラム内の複数の識別子に付加されていくことと定義しているが，注釈を参照する開発者の視点で見た場合は，1 つの識別子の注釈を参照した際に，関連する識別子に付加された注釈も同時に参照できる機構であるといえる．

本研究では，識別子間の関連の種類から，注釈伝播ルールを変数，メソッド，クラスの場合に大きく分類し，以下のように各場合における注釈伝播ルールを提案する．なお，本手法で変数とは，Java 言語でのフィールド，局所変数，仮引数を指し，メソッドは，Java 言語のコンストラクタも含むものとする．またクラスは，Java 言語の参照型であるクラス，列挙型，インタフェース，アノテーションを指すこととする．

- 変数宣言における注釈伝播
  - 変数宣言の型 ⇒ 変数
  - 変数初期化子に出現する識別子 ⇒ 変数
- メソッド宣言における注釈伝播
  - 仮引数・戻り値・例外の型，仮引数の変数 ⇒ メソッド

- 親クラスのオーバーライドされたメソッド ⇒ メソッド
- 同一クラスでオーバーロードされたメソッド ⇒ メソッド

- クラス宣言における注釈伝播
  - 親クラス ⇒ 子クラス

変数宣言における注釈伝播　Java 言語で変数として宣言できるフィールドや局所変数，仮引数の宣言文に出現する型や初期値は，その変数の意味に関連することから，注釈伝播の対象とした．具体的には，変数宣言文中に含まれる変数の型から変数へと注釈が伝播され，また，変数の初期化子（初期値を記述した式）に出現する識別子から変数へと注釈が伝播される．これにより開発者は，型の情報や初期化部分の情報を，注釈伝播によって効果的に調べることができる．

変数の初期化部分から注釈伝播が行われる識別子はたかだか 1 つとした．具体的には初期化子である右辺式について，2 項演算子を含まない項のうち最も最初に評価される項の中で，最後に評価される識別子の DocumentTag だけが伝播される．例えば，初期化部分でコンストラクタ呼び出しのみが行われている場合は，コンストラクタの注釈が伝播され，文字列の連結などで 2 項演算子を含む初期化式では，最初に評価される項からのみ注釈伝播が行われ，後に評価される項は無視される．これにより，変数の初期化部分を開発者が把握する上で効果的な注釈のみを伝播し，多数の注釈が同時に伝播される問題を回避している．

メソッド宣言における注釈伝播　メソッドやコンストラクタの宣言では，メソッドの機能を理解する上で引数や戻り値の情報が重要であると考え，注釈伝播を定義した．具体的には，メソッドの宣言で出現する，戻り値の型・例外の型・仮引数の名前・仮引数の型から，メソッドへと注釈が伝播される．加えて，親クラスのオーバーライドされたメソッドや，同一クラスのオーバーロードされたメソッドからも同様に注釈伝播が行われる．これにより，開発者は，メソッドの引数の型や例外の型，クラス内・クラス間のメソッドの関連を効果的に調べることができる．

クラス宣言における注釈伝播　クラス間の親子関係に従って注釈伝播を行う．具体的には，クラスから，そのクラスの直接の子クラスとなるクラスへと注釈が伝播される．推移的に子クラスとなるクラスには注釈伝播は行われない．これにより開発者は，クラスの役割を調査する上で必要となる直接の親クラスの注釈を，注釈伝播によって効果的に調べることができる．

4　実装

本研究では，DocumentTag を統合開発環境 Eclipse のプラグイン DocumentTag Editor として実装した．開発者は，エディタ上で識別子を選択することで，識別子に対応した注釈と，その識別子と関連のある識別子の注釈を同時に参照することができる．図 2 に DocumentTag Editor を組み込んだ Eclipse のスクリーンショットを示す．図 2 の左下で開かれているビューが本研究で実装した DocumentTag Editor である．テキストエディタ上の DeclarationRelatedPropagation クラスにカーソルを合わせており，DeclarationRelatedPropagation クラスに対応した注釈と，スーパークラスである PropagationRule クラスの注釈が同時に表示されている．

5　実験

DocumentTag Editor を用いて適用実験を行った．学生 4 人に対して，Java 言語で書かれた既存のプログラムの拡張作業を与え，作業時間などを評価した．

具体的には，DocumentTag Editor 自身のソースコードに対して，その機能を拡張する課題を 2 つ用意し，それぞれを課題 a，課題 b として学生 4 人に順序を入れ替えて作業を行わせた．この実験では，DocumentTag の特徴である注釈伝播が保守作業に有効であるかを評価するため，3 章で述べた注釈伝播をすべて有効にした DocumentTag Editor を用いて作業した場合と，関連する識別子への注釈伝播を無効にした DocumentTag Editor を用いて作業した場合の作業時間を比較した．学生に対する課題の割り当てや条件の割り当ては表 1 のように行った．

実験の課題説明に用いた資料は，Web サイトに配備し，DocumentTag Editor のインストールなど実験環境の準備は，Web 上の資料に従って行う

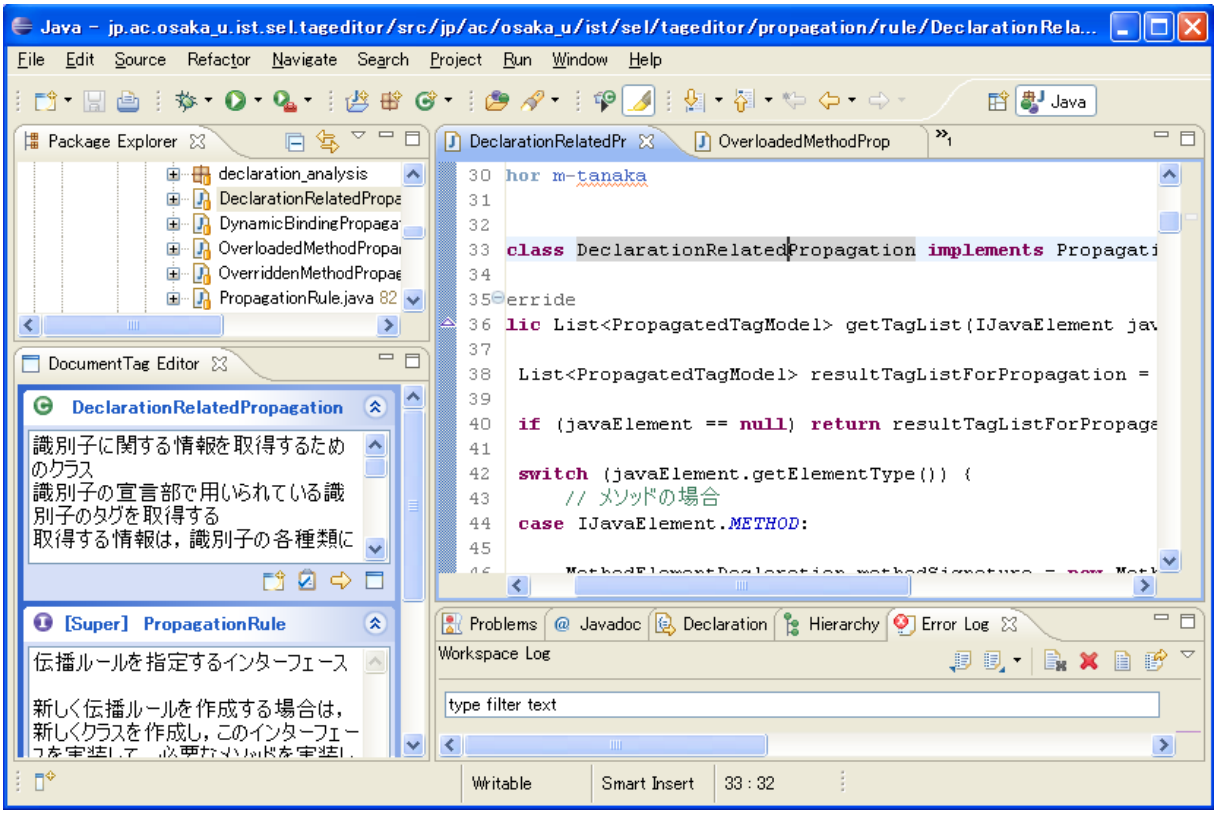

図 2: DocumentTag Editor を組み込んだ Eclipse

表 1: 学生の課題の割り当て表

被験者	課題 1 回目	課題 2 回目
A	課題 a（有効）	課題 b（無効）
B	課題 b（有効）	課題 a（無効）
C	課題 a（無効）	課題 b（有効）
D	課題 b（無効）	課題 a（有効）

こととした．また作業を行うにあたって，Eclipse の操作方法や，課題の文章の意味を確認するための質問に対応するために，実験中は巡回して各被験者に応対した．ソースコードに関する質問に対しては，一切回答を行わないこととした．

5.1 作業時間の比較

表 1 に従って，各課題 a，b を実施したところ，各課題の作業時間は表 2 に示す結果となった．この作業時間は，Eclipse を起動して作業を開始し，課題の動作確認を行い，Eclipse を終了するまでの時間である．

表 2 に示すように，注釈伝播が無効の場合よりも有効の場合の方が，平均作業時間が短くなるという結果が得られた．しかし作業時間にばらつきがあり，統計的に有意差があるとは断定できなかった．

5.2 アンケート結果

本節では課題を終えた学生 4 人に対して行ったアンケート結果について述べる．

アンケートでは，作業課題を終えた学生に，以下に挙げる内容について質問を行った．

- DocumentTag Editor を使用する利点について
  - 課題 a，課題 b，課題全体それぞれについて利点があれば回答する
- DocumentTag Editor の改善点について
- 作業課題を行う中で時間の要した作業について
  - コーディング・課題の理解・プログラム理解・バグ修正・Eclipse 上の操作のうち，時間のかかった順に覚えている範囲で回答する

アンケートの回答を集計したところ，表 3 のような結果となった．表 3(a) から，DocumentTag Editor を用いて関連する注釈も参照できる環境が有効であるという回答が多く得られた．しかし，表 3(b) から，ツールとしての使いやすさに改善の余地があることが分かった．

また，時間のかかった作業として表 3(c)，3(d) と表 3(e)，3(f) とを比較すると，課題 1 回目と課題 2 回はともにプログラム理解に多く時間を要していることが分かった．また，課題 1 回目では課題 2 回目に比べ，とくにコーディングに時間を要する場合があることが分かった．

本研究で行った適用実験では，他識別子からの注釈伝播の機能が無効になっている場合と比べて，注釈伝播を有効にした場合の方が平均作業時間の短縮が見られた．アンケート結果にも挙げられた通り，注釈伝播によってスーパークラスやメソッドの引数の情報を効率的に理解できたことが理由であると考えられる．

被験者の学習効果　被験者 A，D の作業時間に顕著に見られるように，注釈伝播の効果よりも，課題の順序が作業時間に大きく影響する結果となった．これは，Eclipse プラグインというドメインの知識を学習したことによる効果が考えられる [6]．課題 a と課題 b で対象となるソースコードの範囲は，プラグインのメインクラス以外は互いに直接の依存関係を持たないので，対象となったプログラムの知識よりも，コードの書き方を知るために多く時間を要したものと考えられる．実際にアンケート結果では，課題 1 回目でコーディングに多く時間を要した被験者が 2 名見られる．その被験者の回答では，どう書いていいのか分からなかったという理由が 2 名ともに挙げられた．課題 1 回目を終えることによって，コーディングによる拡張方法を学習し，課題 2 回目では大幅にコーディングの時間を短縮したものと考えられる．

6　あとがき

本研究では，開発者が理解した内容を効果的に記録するための付加注釈 DocumentTag を提案し，識別子に対応して注釈を記録できる環境を DocumentTag Editor として実装した．

適用実験として，DocumentTag Editor を用いて，注釈伝播が行われることでプログラム理解が効率的になることを作業時間を基に評価した．その結果，統計的な有意差は確認できなかったが，平均作業時間が短縮される結果となった．

今後の課題としては，構文的な識別子間の関連

表 2: 各被験者ごとの保守作業にかかった時間

被験者	作業時間（分）	
	有効	無効
A（課題 a：有効．課題 b：無効）	131	77
B（課題 b：有効．課題 a：無効）	91	98
C（課題 a：無効．課題 b：有効）	63	87
D（課題 b：無効．課題 a：有効）	45	165
平均時間	82.50	106.75

表 3: アンケート集計結果

項目	回答者
Eclipse 上で識別子を選択するだけで注釈を参照できる点	A,B,C,D
あるクラスを選択するとスーパークラスの仕様も同時に参照できる点	A,D
あるメソッドを選択すると各引数の仕様も同時に参照できる点	C
ソースコードに書き込まずにメモを残せる点	C

(a) DocumentTag Editor を使用する利点

項目	回答者
関連する注釈を閲覧する操作を簡単にしてほしい	B,D
注釈伝播の拡張を別プラグインから拡張できるようにしてほしい	C

(b) DocumentTag Editor の改善点

作業の種類	回答者
プログラム理解	A,D
コーディング	B
バグ修正	C

(c) 課題 1 回目で最も時間を要した作業

作業の種類	回答者
プログラム理解	C
コーディング	A
課題の理解	D
バグ修正	B

(d) 課題 1 回目で 2 番目に時間を要した作業

作業の種類	回答者
プログラム理解	A,C,D
バグ修正	B

(e) 課題 2 回目で最も時間を要した作業

作業の種類	回答者
プログラム理解	B
バグ修正	A
課題の理解	D

(f) 課題 2 回目で 2 番目に時間を要した作業

だけでなく，より高度な注釈伝播の検討がある．たとえば，デザインパターン検出ツール [11] を併用し，デザインパターン上で関連のあるクラスやメソッドについての注釈を伝播する手法が考えられる．また，定数伝播 [2] と同様にデータフロー関係に従って注釈伝播を行うことにより，値の説明を意味した注釈をデバッグに利用できるのではないかと考えられる．このように注釈伝播ルールを拡張すると，1 つの識別子に多数の注釈が伝播しうるため，複数の注釈を要約あるいは順位づけし，効果的に開発者に提示することも重要な課題である．

謝辞

本研究は，日本学術振興会科学研究費補助金若手研究（スタートアップ）（課題番号:19800021）および日本学術振興会科学研究費補助金基盤研究（A）（課題番号:17200001）の助成を得た．

参考文献

[1] P. Anderson, T. Reps, and T. Teitelbaum. Design and implementation of a fine-grained software inspection tool. *IEEE Trans. Softw. Eng.*, Vol. 29, No. 8, pp. 721–733, 2003.

[2] S. Blazy and P. Facon. Partial evaluation as an aid to the comprehension of fortran programs. In *Proceedings of the 2nd IEEE Workshop on Program Comprehension (IWPC '93)*, pp. 46–54, 1993.

[3] J. Bloch. *Effective Java (2nd Edition)*. Prentice Hall, 2008.

[4] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns*. Chapter 5.1 Tie Code and Questions. Square Bracket Associates, 2008. `http://www.iam.unibe.ch/~scg/OORP/`.

[5] M. Desmond, M.-A. Storey, and C. Exton. Fluid source code views for just in-time comprehension. In *Workshop on Software Engineering Properties of Languages and Aspect Technologies (SPLAT '06)*, 2006.

[6] T. Fritz, G. C. Murphy, and E. Hill. Does a programmer's activity indicate knowledge of code? In *Proceedings of the the 6th joint meeting of the 11th European Software Engineering Conference and the 15th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07)*, pp. 341–350, 2007.

[7] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In *Proceedings of the 4th International Conference on Aspect-Oriented Software Development (AOSD '05)*, pp. 159–168, 2005.

[8] M. Lejter, S. Meyers, and S. P. Reiss. Support for maintaining object-oriented programs. *IEEE Trans. Softw. Eng.*, Vol. 18, No. 12, pp. 1045–1052, 1992.

[9] C. Oezbek and L. Prechelt. JTourBus: Simplifying program understanding by documentation that provides tours through the source code. In *Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM '07)*, pp. 64–73, 2007.

[10] M.-A. Storey, L.-T. Cheng, J. Singer, M. Muller, D. Myers, and J. Ryall. How programmers can turn comments into waypoints for code navigation. In *Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM '07)*, pp. 265–274, 2007.

[11] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis. Design pattern detection using similarity scoring. *IEEE Trans. Softw. Eng.*, Vol. 32, No. 11, pp. 896–909, 2006.

[12] N. Wilde and R. Huitt. Maintenance support for object-oriented programs. *IEEE Trans. Softw. Eng.*, Vol. 18, No. 12, pp. 1038–1044, 1992.