メソッド抽出リファクタリングが行われるメソッドの特徴調査

後藤 祥　吉田 則裕　藤原 賢二　崔 恩瀞　井上 克郎

メソッド抽出とは，既存のメソッドの一部を新たにメソッドとして抽出する作業のことである．メソッド抽出は，長すぎるメソッドや，凝集性の低いメソッドを分割するために有効であるとされているが，実際に開発者がどのようなメソッドを対象にしているかは調査されていない．しかし，メソッド抽出作業の支援を行うためには，開発者がどのようなメソッドを抽出の対象としているかを定量的に調査する必要がある．本研究では，オープンソースソフトウェアから，抽出が行われたメソッドと行われなかったメソッドの間で，サイズと凝集度に違いがあるか調査した．調査の結果，多くの場合に有意差があることが確認できた．

"Extract Method" is a refactoring pattern that extracts a part of an existing method as a new method. Although extract method refactoring is an effective way to decompose long and non-cohesive methods in general, how developers choose methods for "Extract Method" refactoring is still unexamined. For supporting this refactoring, the investigation of it is necessary. In this study, we investigated the differences of the size and cohesion of methods between refactored methods and not-refactored methods in open source software. The result shows significant deliverances in the most cases.

1　はじめに

リファクタリングとは，ソフトウェアの外的な振る舞いを保ったまま内部構造を整理する作業のことであり，プログラムの保守性や可読性の向上を目的として行われる [1]．Fowler は，様々なリファクタリングパターンをまとめており，それぞれのパターンについてその用途や手順を記述している [1]．リファクタリングはソフトウェアの保守性を向上させるが，大規模なソースコードの中からリファクタリングの対象を特定する作業は大きな労力を要する．そのため，リファクタリング作業者を支援するための手法やツールが多く提案されている [5] [9]．

近年，開発者が行うリファクタリングの実態を調査し，その結果に基づいて支援手法を考案する研究が行われている [6] [10]．既存研究では，各リファクタリングパターンが適用される頻度や Eclipse のリファクタリング機能の利用状況などが調査されているが，リファクタリング対象となったソースコードが持つ特徴の定量的な調査は確認されていない．一般的に，凝集度が低い，もしくはサイズの大きいといった特徴をもつソースコードがリファクタリングの対象になるとされているが，実際の開発プロジェクトにおけるリファクタリングが，それら尺度に基づいて行われているかどうか明らかではない．そのため，それら尺度に基づいてリファクタリング対象を推薦すべきであるかどうか判断することが難しい．

本研究では，頻繁に行われるリファクタリングの 1 つであるメソッド抽出 [6] に着目し，抽出対象となったメソッドの特徴を調査した．具体的には，抽出が行われたメソッドのサイズおよび凝集度を計測し，抽

An Investigation into the Characteristics of Methods for Extract Method Refactoring.

Akira Goto, Eunjong Choi, Katsuro Inoue, 大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University.

Norihiro Yoshida, Kenji Fujiwara, 奈良先端科学技術大学院大学情報科学研究科, Graduate School of Information Science, Nara Institute of Science and Technology.

コンピュータソフトウェア, Vol.31, No.3 (2014), pp.318–324.
[研究論文 (レター)] 2013 年 12 月 19 日受付.

出が行われなかったメソッドとの比較を行った．比較の結果，抽出が行われたメソッドと行われなかったメソッドでは，メソッドのサイズおよび凝集度に有意差が存在することがわかった．

2 背景

2.1 メソッド抽出

メソッド抽出は，既存のメソッドの一部を抽出し，新たにメソッドとして定義するリファクタリングである．図 1 にメソッド抽出の例を示す．図 1 の例において，2 つの文が抽出され，新たに *printDetails* メソッドとして定義されている．抽出元の *printOwing* メソッドでは，抽出部分のソースコードが *printDetails* メソッドの呼び出し文に置換されている．

メソッド抽出は，長すぎるメソッドや，凝集性の低いメソッドの分割を目的として行われるとされている．しかし，実際に長いメソッドが抽出の対象となっているのかどうかや，数値的な尺度などを定量的に調査した研究は著者らの知る限り存在しない．

2.2 藤原らのリファクタリング検出ツール

リファクタリング検出ツールとは，ソフトウェアの開発履歴から，どのリファクタリングパターンが，どの部分に適用されたかを検出するためのツールである．リファクタリング検出ツールを用いて，リファクタリングの事例を収集することで，リファクタリングの頻度やリファクタリング対象とされるソースコードの特徴など様々な調査を行うことが可能となる．

藤原らは，構文情報を付加したリポジトリを利用して，メソッド抽出リファクタリングの検出を行うツールを提案している [2]．藤原らのツールでは，コミット間での変更情報を元にメソッド抽出が行われたと推定される箇所を特定し，抽出元のメソッド，メソッド抽出によって新たに作成されたメソッド，抽出元のコード片と新たに作成されたメソッドの類似度を出力する．この類似度は 0 から 1 の値であり，類似度が高いほどメソッド抽出が行われた可能性が高いことを意味している．

藤原らは，リファクタリング検出ツールである UMLDiff [12] と結果を比較することでツールの精度

```
void printOwing(){
  printBanner();

  //print details
  System.out.println("name: " + _name);
  System.out.println("amount: " + getOutstanding());
}
```

```
void printOwing(){
  printBanner();
  printDetails(getOutstanding());
}

void printDetails(double outstanding){
  System.out.println("name: " + _name);
  System.out.println("amount: " + outstanding);
}
```

図 1 メソッド抽出リファクタリング [1]

の評価を行っている．評価によって，藤原らのツールは Presicion が 0.96，Recall が 0.86 と高い精度でメソッド抽出リファクタリングの検出が可能であることを示している．また，藤原らのツールは大規模な履歴に対しても，容易に適用可能であるという利点がある．本研究では，比較調査のために多くのリファクタリング事例が必要であるため，藤原らのツールを用いてメソッド抽出事例の収集を行った．

3 調査方法

3.1 調査する特徴

メソッド抽出は長すぎるメソッドや，凝集性の低いメソッドを分割するために有効であるとされている．本研究では，メソッドのサイズと凝集性を，それぞれメソッドの文数と凝集度という尺度で計測を行い，抽出対象となったメソッドが持つ特徴を調査する．文数と凝集度について，具体的にどのようにして計測するか説明する．

1. NOS(Number Of Statements)

   メソッドのサイズの計測方法については，単純に行数を測るものや，空白行やコメントを除いて測る方法などが存在する．メソッド抽出はメソッド中の文に対して行われるリファクタリングであり，メソッド中のコメントや空白行の数とは大きな関連がないと思われる．そのため，本研究では，メソッドのサイズをコメントと空白行を除去

表 1 調査対象ソフトウェア (括弧内は凝集度が計測可能なメソッドの内数)

ソフトウェア	調査対象期間	メソッド抽出事例数	比較対象メソッド数 (抽出無)
jEdit	1998/9/27 - 2012/1/30	490(286)	490(286)
Ant	2000/1/13 - 2012/5/23	659(302)	659(302)
ArgoUML	1998/1/27 - 2011/12/15	704(322)	704(322)

したメソッドの本体中の文数とした.

2. 凝集度

凝集度とは，モジュールが機能的にまとまったものかどうかを表す度合である [8]. 凝集度を計測するためのメトリクスは複数提案されているが，本研究では計測対象がメソッドであるため，メソッド単位の凝集度メトリクスであるスライスベースの凝集度メトリクス [11] を用いる．スライスベースの凝集度メトリクスは，Weiser によって 5 つ提案されており，そのうち Tightness, Coverage，Overlap の 3 つがメソッドの凝集度を計測するのに有用であることを Ott らが実験によって示している [7]. 本研究ではこれら 3 つのメトリクスを用いてメソッドの凝集度を計測する．それぞれのメトリクスの定義を以下に示す．式において，M をメソッド，$length(M)$ を M の文の数，V_o を M における出力変数の集合，SL_x を変数 x を起点にした後ろ向きスライス，SL_{int} を V_o 中の全変数に対する後ろ向きスライスの積集合とする．

$$Tightness(M) = \frac{|SL_{int}|}{length(M)}$$

$$Coverage(M) = \frac{1}{|V_o|}\sum_{x\in V_o}\frac{|SL_x|}{length(M)}$$

$$Overlap(M) = \frac{1}{|V_o|}\sum_{x\in V_o}\frac{|SL_{int}|}{|SL_x|}$$

メトリクスの計算に必要となるメソッドの出力変数 V_o は，Tsantalis らの定義に従い，メソッドの返り値と，メソッドの本体とスコープが一致する変数とした [9]. これらのメトリクスは，メソッドの出力変数に基づいて凝集度を計算するものであり，スライスの起点となる変数が存在しないメソッドに対しては，凝集度を計算することができない.

3.2 調査対象

本研究の調査対象として jEdit, Ant，ArgoUML の 3 つのオープンソースソフトウェアを選択した．調査対象の詳細を表 1 に示す.

メソッド抽出事例の検出は，調査対象のソフトウェアに藤原らのツールを適用することで行った†1．表 1 において，メソッド抽出事例数は，藤原らのツールを使用して検出された調査対象期間内で行われたメソッド抽出のうち，テストメソッドを除いたメソッドの数を表している.

表 1 のとおり，比較対象メソッドはメソッド抽出事例と同じ数用意した．括弧内の数値は，メソッド抽出事例のうち凝集度を計算可能なメソッドの数を示している†2．比較対象のメソッドは，まずランダムでリビジョンを選択し，そのリビジョンに存在する全メソッドの中から，次のコミットでメソッド抽出が行われなかったメソッドをランダムで 1 つ選択するという処理を繰り返し行うことで収集した.

4 調査結果と考察

各対象ソフトウェアについて，計測した NOS と凝集度の分布を表した箱ひげ図を図 2, 3 に示す．以降の結果において,「抽出有」はメソッド抽出が行われたメソッドに対する結果を意味しており,「抽出無」はメソッド抽出が行われなかったメソッドに対する結果

†1 藤原らの研究 [2] において，類似度の閾値が 0.3 の時，検出精度が最も高いと述べられているため，本研究でも同一の閾値を利用し，出力結果から閾値以上の類似度をもつものを調査対象として利用した.

†2 各ソフトウェアのメソッド抽出事例と比較対象メソッド間において，凝集度を計算可能なメソッド (計算不可能なメソッド) の数を同数に揃えた.

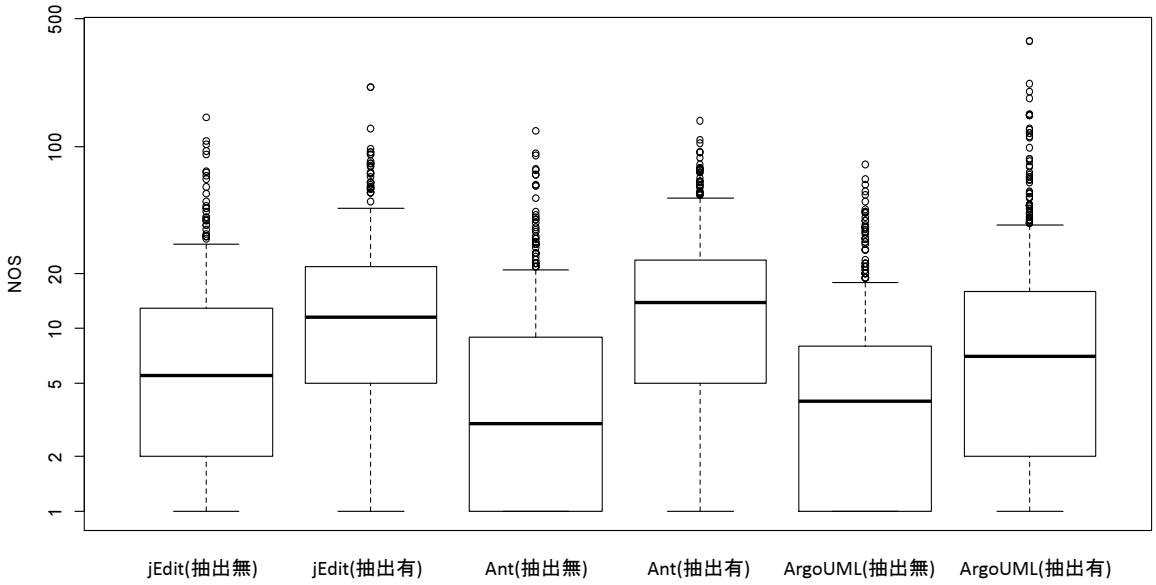


図 2 NOS 分布

を意味している．また図 3 において，T, C, O はそれぞれ Tightness, Coverage, Overlap の 3 つのメトリクスを意味している．

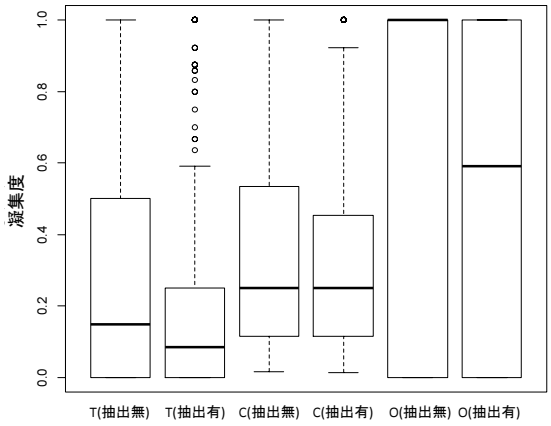
抽出が行われたメソッドと行われなかったメソッドについて，計測した特徴に有意差が存在するかどうか確認するため，マンホイットニーの U 検定を用いて検定を行った．表 2，3，4 に，NOS と凝集度の中央値と，検定の結果得られた p 値を示す．

表 2，3，4 より，jEdit における Coverage と，ArgoUML における Overlap 以外の全てにおいて，有意水準 0.05 で有意差があるという結果になった．これらの結果から，今回対象としたソフトウェアにおいては，メソッドの NOS が多く，凝集度が低いメソッドが抽出の対象となっていると言える．図 2 を見ると，NOS については，抽出対象となっているメソッドの多くは，約 5 文から 20 文程度であることがわかった．

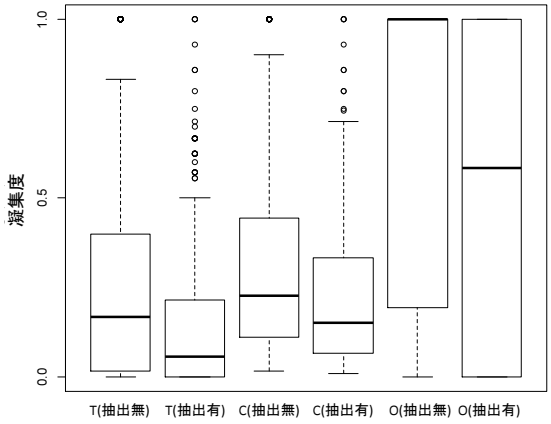
次に，図 3 から，各凝集度メトリクスの結果について考察する．Tightness は全てのソフトウェアにおいて有意差がみられた．また抽出が行われたメソッドについては，中央値が全て 0.1 以下であり，jEdit と Ant では 0.2 以下の値を持つメソッドがほとんどであった．Coverage は Ant と ArgoUML では有意差があったが，jEdit では有意差が見られなかった．Coverage については，対象ソフトウェア間で結果に差が見られたため，今回対象とした以外のソフトウェアに対しても調査を行い，結果を確認する必要があると思われる．Overlap についても，ArgoUML のみ有意差がなかったため，Coverage と同様に，他のソフトウェアでどのような結果になるか調査を行う必要があると考えられる．さらに Overlap メトリクスは，Tightness と Coverage とは値の分布が大きく異なっているという特徴が見られた．これは，Overlap メトリクスは起点になる変数が 1 つしか存在しない場合は必ず値が 1 になることが理由と考えられ，実際に抽出が行われなかったメソッドに対する結果では，半数以上のメソッドが Overlap の値が 1 になっている．

調査の結果から，抽出が行われたメソッドと行われなかったメソッドの NOS と凝集度は，多くの場合有意差があり，抽出が行われたメソッドの方が NOS が多く，凝集度が低いことがわかった．

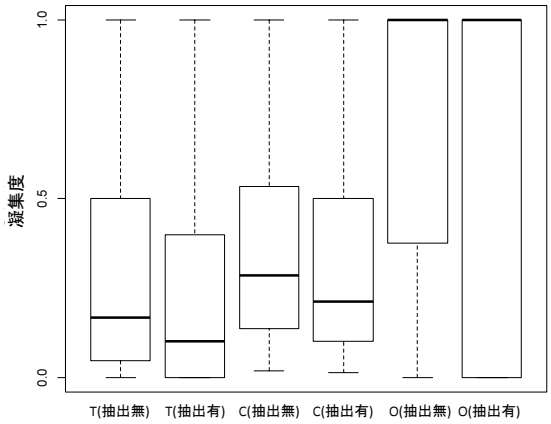
最後に，本研究で得られた結果の妥当性について述べる．まず，使用したメトリクスについてである．本研究では，凝集度を測るメトリクスとして，メソッド単位のメトリクスであることや既存研究 [7] における実験で有用性が示されているという理由で，スライスベースの凝集度メトリクスを選択した．しかし，



(a) jEdit



(b) Ant



(c) ArgoUML

図 3　メソッドの凝集度分布

表 2　比較結果 (jEdit)

	中央値	p 値
NOS(抽出無)	5.5	p<0.05
NOS(抽出有)	11.5	
Tightness(抽出無)	0.148	p<0.05
Tightness(抽出有)	0.083	
Coverage(抽出無)	0.250	0.05<p
Coverage(抽出有)	0.250	
Overlap(抽出無)	1.00	p<0.05
Overlap(抽出有)	0.592	

表 3　比較結果 (Ant)

	中央値	p 値
NOS(抽出無)	3.0	p<0.05
NOS(抽出有)	14.0	
Tightness(抽出無)	0.167	p<0.05
Tightness(抽出有)	0.056	
Coverage(抽出無)	0.227	p<0.05
Coverage(抽出有)	0.150	
Overlap(抽出無)	1.000	p<0.05
Overlap(抽出有)	0.583	

表 4　比較結果 (ArgoUML)

	中央値	p 値
NOS(抽出無)	4.0	p<0.05
NOS(抽出有)	7.0	
Tightness(抽出無)	0.168	p<0.05
Tightness(抽出有)	0.100	
Coverage(抽出無)	0.286	p<0.05
Coverage(抽出有)	0.211	
Overlap(抽出無)	1.000	0.05<p
Overlap(抽出有)	1.000	

スライスベースの凝集度メトリクスは，処理の一部がスライスに含まれない場合など，正しく凝集度を算出できない場合がある．そのため，今後の他の凝集度メトリクスを用いてメソッドの特徴を調査する必要があると考えられる．他のメソッド単位の凝集度メトリク

スとしては，Lakhotia が提案している変数の依存関係とそれらのルールに基づくメトリクス [3] や，三宅らが提案しているコードブロック間での変数の共有に基づくメトリクスなどが挙げられる [4].

次に，リファクタリング検出ツールについてである．本研究の結果がリファクタリング検出ツールに依存したものではないことを確認するため，本研究で使用した藤原らの検出ツールと UMLDiff [12] の検出結果の間に特徴の偏りがないかを調べた．藤原らのツールによって検出されたメソッドと，藤原らのツールで検出されなかったが UMLDiff で検出されたメソッドに対して，それぞれ特徴を計測し，特徴の値に差が存在するかマンホイットニーの U 検定を用いて検定を行った．検定の結果，全ての特徴について有意水準 0.05 で有意差なしという結果が得られた．そのため，これらの結果はリファクタリング検出ツールに依存したものではないと考えられる．

最後に，比較対象メソッドの抽出についてである．3.2 節において，次のコミットにおいてメソッド抽出が行われなかったメソッドを選択したが，次のコミット以降でメソッド抽出が行われた場合を考慮できていない．今後，このような場合について調査を行う必要がある．

5 まとめ

本研究では，メソッド抽出が行われたメソッドが持つ特徴の定量的調査を行った．調査においては，オープンソースソフトウェアから収集した，抽出が行われたメソッドと行われなかったメソッドに対して，NOS と凝集度の計測を行い，それらの比較を行った．比較の結果，抽出が行われたメソッドは NOS が多く，凝集度が低いことがわかった．

今後の課題としては，対象ソフトウェアと計測する特徴を追加した大規模な調査を行うことが挙げられる．その調査を行う際に，リファクタリングに関する規約の有無が調査結果に与える影響をあわせて確認したいと考えている．また，開発者が NOS が大きいメソッドをメソッド抽出の対象としているのではなく，均一の確率で文を選択し，その文が含まれるメソッドをメソッド抽出の対象と考えた結果，NOS に有意差が生じた可能性がある．今後，ランダムで選択した文を含むメソッドを対象としたときの NOS と比較しても，実際にメソッド抽出の対象となったメソッドの NOS が十分に大きいか確認する必要がある．これら調査を行ったあと，メトリクスを利用した機械学習を用いることで，メソッド抽出の対象となるメソッドを推薦する手法を実現したいと考えている．

謝辞 本研究は JSPS 科研費 25220003，26730036 の助成を受けた．

参 考 文 献

[1] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 1999.
[2] 藤原賢二, 吉田則裕, 飯田元: ソフトウェアリポジトリを対象とした細粒度リファクタリング検出, ソフトウェア工学の基礎ワークショップ, 2013, pp. 101–106.
[3] Lakhotia, A.: Rule-based approach to computing module cohesion, in *Proc. of ICSE*, 1993, pp. 35–44.
[4] 三宅達也, 肥後芳樹, 井上克郎: メソッド抽出の必要性を評価するソフトウェアメトリックスの提案, 信学論, Vol. J92-D, No. 7(2009), pp. 1071–1073.
[5] Murphy-hill, E. and Black, A. P.: Breaking the barriers to successful refactoring: observations and tools for extract method, in *Proc. of ICSE*, 2008, pp. 421–430.
[6] Murphy-Hill, E., Parnin, C. and Black, A. P.: How we refactor, and how we know it, in *Proc. of ICSE*, 2009, pp. 287–297.
[7] Ott, L. M. and Thuss, J. J.: Slice Based Metrics for Estimating Cohesion, in *Proc. of METRICS*, 1993, pp. 71–81.
[8] Stevens, W. P., Myers, G. J. and Constatine, L. L.: Structured design, *IBM Syst.J.*, Vol. 13, No. 2(1974), pp. 115–139.
[9] Tsantalis, N. and Chatzigeorgiou, A.: Identification of extract method refactoring opportunities for the decomposition of methods, *Journal of Systems and Software*, Vol. 84, No. 10(2011), pp. 1757–1782.
[10] Vakilian, M., Chen, N., Negara, S., Rajkumar, B. A., Bailey, B. P. and Johnson, R. E.: Use, disuse, and misuse of automated refactorings, in *Proc. of ICSE*, 2012, pp. 233–243.
[11] Weiser, M.: Program slicing, in *Proc. of ICSE*, 1981, pp. 439–449.
[12] Xing, Z. and Stroulia, E.: Refactoring Detection based on UMLDiff Change-Facts Queries, in *Proc. of WCRE*, 2006, pp. 263–274.

後藤　祥

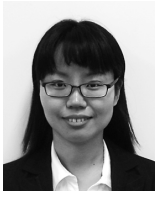
2012 年大阪大学基礎工学部情報科学科卒業．2014 年同大学大学院情報科学研究科博士前期課程修了．現在，新日鉄住金ソリューションズ株式会社に勤務．在学中，リファクタリング支援の研究に従事．

吉田則裕

2004 年九州工業大学情報工学部知能情報工学科卒業．2009 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員 (PD)．2010 年奈良先端科学技術大学院大学情報科学研究科助教．2014 年より名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．博士 (情報科学)．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

藤原賢二

2010 年大阪府立工業高等専門学校総合工学システム専攻修了．2012 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在同大学情報科学研究科博士後期課程に在籍．修士 (工学)．リファクタリングの適用履歴分析，プログラミング教育支援に興味を持つ．



崔　恩瀞

2012 年大阪大学大学院情報科学研究科博士前期課程修了．現在同大学大学院情報科学研究科博士後期課程に在籍．修士 (情報科学)．コードクローン管理手法やリファクタリング支援手法に関する研究に従事．

井上克郎

1984 年大阪大学大学院基礎工学研究科博士後期課程修了 (工学博士)．同年，大阪大学基礎工学部情報工学科助手．1984〜1986 年，ハワイ大学マノア校コンピュータサイエンス学科助教授．1991 大阪大学基礎工学部助教授．1995 年同学部教授．2002 年大阪大学大学院情報科学研究科教授．2011 年 8 月より大阪大学大学院情報科学研究科研究科長．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．