早稲田大学大学院 基幹理工学研究科

博 士 論 文 概 要

論 文 題 目

A Study on Source Code Reengineering Frameworks Supporting Multiple Programming Languages

複数のプログラミング言語に対応するソースコードリエンジニアリングフレームワークに関する研究

申 請 者

坂本	一憲
Kazunori	SAKAMOTO

情報理工学専攻
高信頼ソフトウェア工学研究

2012 年 12 月
（受理申請する部科主任会開催年月を記入）

概要本文１ページ目（1ページあたり明朝体 11 ポイント ３６文字×３７行で３ページ以内）

Program source code reengineering is one of key technologies in software development for improving software quality with low costs. For example, software metrics are important indicators for assessing software quality which are acquired by analyzing source code, test coverage is also an important indicator for assessing test quality which is acquired by transforming source code to store execution logs and an AOP (aspect-oriented programming) processor makes source code high modularity by weaving aspects into source code. Such means are based on source code reengineering and many kinds of reengineering tools exist.

Programming languages have become more diversified. There are many paradigms of programming languages: statically-typed / dynamically-typed, imperative / declarative, functional / logic and object-oriented / aspect-oriented / context-oriented. Software development using multiple programming languages is required because each programming language has an area of specialty and programmers choose suitable programming languages for each project. In particular, web applications are based on the client-server model and usually use three programming languages: HTML, JavaScript and another programming language. The development of web applications has been more and more frequent with the popularization of the Internet. The tools therefore are required to support multiple programming languages for such development.

However, there are two problems in existing tools. (1) It requires high costs to develop tools supporting multiple programming languages due to the variety of programming languages. Many tools thus support few programming languages and the users receive only little benefits from the tools. (2) There are differences between tools supporting different programming languages because each tool is developed for specific programming languages and supports few programming languages. For example, measurement tools of test coverage, Cobertura supports only Java and Coverage.py supports only Python. However, there is no tool which measures test coverage for TypeScript, thus, programmers cannot measure test coverage for TypeScript. Moreover, whereas Cobertura supports statement coverage and decision coverage, Statement coverage for Python supports only statement coverage. It is difficult to measure decision coverage for web applications using Java and Python. Therefore, means are required to reduce development costs of tools supporting multiple programming languages and to reduce inconsistency between tools.

To solve the problems, this thesis proposes a novel two frameworks called OCCF (Open Code Coverage Framework) and UNICOEN (Unified Code Reengineering Framework). OCCF is a consistent, flexible and complete framework for measuring test coverage supporting multiple programming languages. UNICOEN is a framework developed by generalizing OCCF for reengineering source code supporting multiple programming languages.

The thesis consists of 6 chapters.

Chapter 1 “Introduction” states the objective this research with research background. This chapter also defines the research area of the thesis by referring related works.

Chapter 2 “Software Development Using Multiple Programming Languages” describes the situation where software development using multiple programming languages is required. This chapter summarizes programming language paradigms and shows the relation of application types and suitable programming languages. The chapter also describes the importance of researches to overcome the variety of programming languages.

Chapter 3 “OCCF (Open Code Coverage Framework): A Consistent and Flexible Framework for Measuring Test Coverage Supporting Multiple Programming Languages” describes the proposed framework called OCCF. Existing measurement tools have four problems. (1) Developing measurement tools supporting multiple programming languages requires high costs. There is no free measurement tool for legacy and new programming languages such as COBOL and Kotlin. Such situation makes it difficult to maintain and introduce legacy and new programming languages. (2) Existing tools, which support different programming languages, measure also different coverage criteria. The difference makes it difficult to introduce test coverage for software using multiple programming languages such as web applications. (3) It is difficult customize existing measurement tools for utilizing special coverage criteria. The effort to add new coverage criteria in the tools is significant. (4) Some existing tools insert measurement code into compiled binary file to measure test coverage. However, this way misses dead code because the compiler optimization removes any dead code. Users therefore cannot notice dead code from the result of test coverage. To overcome the problems, OCCF reduces development costs of measurement tools by providing reusable architecture and code. The reusable architecture and code help users to implement tools supporting consistent coverage criteria: statement coverage, decision coverage, condition coverage, condition/decision coverage. OCCF also provides hot spots to customize coverage criteria. Moreover, OCCF inserts the measurement code into source code and allows user to notice the existence of dead code. The effectiveness of OCCF is evaluated by experiments in the chapter. As a result of the experiments and the application, OCCF alleviates problems.

Chapter 4 “An Application of OCCF for Minimizing Test Cases Based on Test Coverage” describes a tool developed with OCCF. The tool minimizes test cases by judging whether a test case is duplicated with other test cases based on test coverage. The tool considers that the test case which executes the same elements of production code executed by other test cases is duplicated in terms of suitable test coverage to measure such elements. This chapter also describes the evaluation of the tool through applications in existing open source software. As a result of the application development, this chapter shows the effectiveness of OCCF.

Chapter 5 “UNICOEN (Unified Code Reengineering Framework): A Unified Framework for Code Reengineering Supporting Multiple Programming Languages” describes the proposed framework called UNICOEN. UNICOEN generalizes OCCF to support not only tools for measuring test coverage and also tools for analyzing and transforming source code. Existing tools have two problems. (1) Although some previous works propose frameworks for developing tools which reengineers source code, there is no framework which provides common language model

with API of both analysis and transform and which allows users to extend supports of programming languages. Therefore, some programming languages have no tool for analyzing or transforming. For example, Lint, JSLint and Pyklint are static analyzers for finding bugs. They support C, JavaScript and Python respectively. However, no tool supports Ruby. (2) Many programmers use many programming languages. In particular, most developers of open source software use more than three programming languages. However, there are differences between existing tools, which supports different programming languages. For example, AspectJ supporting Java and AOJS supporting JavaScript are AOP language processors. AspectJ and AOJS support different join points and provide different grammars for writing aspects. Users therefore should learn the both grammars to introduce AOP in development of web applications using Java and JavaScript. To solve the problems, UNICOEN provides common language model, called UCM (Unified Code Model). UCM is developed by adding elements of seven programming languages: C, C#, Java, Visual Basic, JavaScript, Python and Ruby. UCM thus can represent source code of the seven programming languages in common model similarly. UNICOEN defines UCM in terms of syntaxes. UNICOEN assumes that similar syntax have similar semantics between different programming languages. Although UNICOEN cannot interpret semantics of all elements from syntax completely, most elements can be distinguished and can be interpreted because the assumption is valid for most cases. This feature dramatically reduces costs for adding new supports of programming languages compared to other frameworks which interpret semantics. UNICOEN provides two kinds of API for adding supports of programming languages in UNICOEN and for developing tools for reengineering source code. The API provides reusable code and useful methods similar to LINQ in .NET framework. UNICOEN therefore reduces costs for developing tools by providing such API for reengineering source code. The chapter also describes measurement tool of cyclomatic complexity with UNICOEN for evaluating UNICOEN. In comparison to a similar tool for Ruby, the tool with UNICOEN has less lines of code and supports more programming languages. Moreover, the chapter describes evaluations in comparison to programming languages processors and other frameworks. The evaluations indicate that UNICOEN reduces costs to add new programming language supports and to develop tools supporting multiple programming languages. As a result, OCCF alleviates problems.

Chapter 6 “An Application of UNICOEN for Aspect-Oriented Programming Processors” describes a tool developed with UNICOEN. The tool is an AOP processor supporting seven programming languages supported by UNICOEN, called UniAspect. UniAspect provides four join points for supported programming languages similarly: call, execution, get and set. UniAspect also provides aspects which contain language-independent pointcuts and a set of advices written in each programming languages to weave. UniAspect language-independently weaves the aspects into source code by modifying objects on UCM. As a result of UniAspect development, this chapter shows the effectiveness of UNICOEN.

Chapter 7 “Conclusions” concludes the thesis and explains future works.

早稲田大学　博士（工学）　学位申請　研究業績書

氏 名　　坂本　一憲　　印

（2012 年 11 月　現在）

種 類 別	題名、　発表・発行掲載誌名、　発表・発行年月、　連名者（申請者含む）
1.論文 (1) ○	坂本一憲，大橋昭，太田大地，鷲崎弘宜，深澤良彰，″UNICOEN: 複数プログラミング言語対応のソースコード処理フレームワーク,″ 情報処理学会論文誌，54-2，15 pages，2013.（掲載決定）
(2) ○	坂本一憲，大橋昭，鷲崎弘宜，深澤良彰，″コンピュータプレイヤーのプログラム作成を通して競い合うゲームプラットフォームの開発を支援するフレームワーク,″ 電子情報通信学会論文誌，Vol.J95-D，No.3，pp.412-424，March 2012.
(3)	鷲崎弘宜，坂本一憲，大杉直樹 他 7 名，″デザインパターンへのソフトウェア工学的取り組み,″ コンピュータソフトウェア，Vol.29，No.1，pp.130-146，January 2012.
(4) ○	Kazunori Sakamoto, Fuyuki Ishikawa, Hironori Washizaki, and Yoshiaki Fukazawa, ″Open Code Coverage Framework: A Framework for Consistent, Flexible and Complete Measurement of Test Coverage Supporting Multiple Programming Languages,″ IEICE Transactions on Information and Systems, Vol.E94-D, No.12, pp.2418-2430, December 2011.
2.講演 (国際会議) (1)	Ryushi Shiohama, Hironori Washizaki, Shin Kuboaki, Kazunori Sakamoto, and Yoshiaki Fukazawa, ″Estimate of the appropriate iteration length in agile development by conducting simulation,″ pp.41-50, Agile, August 2012.
(2)	Reisha Humaira, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa, ″Towards a Unified Source Code Measurement Framework Supporting Multiple Programming Languages,″ The 24th International Conference on Software Engineering and Knowledge Engineering, pp.480-485, July 2012.
(3)	Akira Ohashi, Kazunori Sakamoto, Tomoyuki Kamiya et al. (4 other authors), ″UniAspect: A Language-Independent Aspect-Oriented Programming Framework,″ The 2nd Workshop on Modularity in Systems Software, pp.39-44, March 2012.
(4)	Yuto Nakamura, Kazunori Sakamoto, Kiyohisa Inoue, Hironori Washizaki, and Yoshiaki Fukazawa, ″Evaluation of Understandability of UML Class Diagrams by Using Word Similarity,″ The 6th International Conference on Software Process and Product Measurement, pp.178-187, November 2011.
(5) ○	Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa, ″Open Code Coverage Framework: A Consistent and Flexible Framework for Measuring Test Coverage Supporting Multiple Programming Languages,″ 10th International Conference on Quality Software, pp.262-269, July 2010.
(6) ○	Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa, ″Reporting the Implementation of a Framework for Measuring Test Coverage based on Design Patterns,″ 3rd International Workshop on Software Patterns and Quality, pp.16-20, 2009.

早稲田大学　博士（工学）　学位申請　研究業績書

種 類 別	題名、　発表・発行掲載誌名、　発表・発行年月、　連名者（申請者含む）
2.講演（シンポジウム）(7) ○	坂本一憲, 海津智宏, 波村大悟, 鷲崎弘宜, 深澤良彰, ″Web アプリの動的部分に着目したグレーボックス統合テストとテンプレート変数カバレッジの提案,″ 第 19 回 ソフトウェア工学の基礎ワークショップ論文集, 8 pages, December 2012. （掲載決定、IEEE CS Japan Chapter Young Researcher Award）
(8)	青井翔平, 坂本一憲, 鷲崎弘宜, 深澤良彰, ″DePoT: Web アプリケーションテストにおけるテストコード自動生成テスティングフレームワーク,″ 第 19 回 ソフトウェア工学の基礎ワークショップ論文集, 8 pages, December 2012. （掲載決定）
(9) ○	坂本一憲, 大橋昭, 太田大地, 鷲崎弘宜, 深澤良彰, ″UNICOEN: 複数プログラミング言語対応のソースコード処理フレームワーク,″ ソフトウェアエンジニアリングシンポジウム 2012 論文集, pp.1-8, August 2012.
(10)	Reisha Humaira, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa, ″A Unified Source Code Measurement Tool Supporting Multiple Programming Languages,″ ウィンターワークショップ 2012, pp.5-6, January 2012.
(11)	神谷知行, 坂本一憲, 大橋昭, 鷲崎弘宜, 深澤良彰, ″複数のプログラミング言語に対応する拡張可能なリファクタリングエンジン″, ウィンターワークショップ 2012, pp.23-24, January 2012.
(12) ○	Kazunori Sakamoto, Akira Ohashi, Masaya Shimizu, Shuhei Takahashi, Shinich Murakami, Hironori Washizaki, and Yoshiaki Fukazawa, ″A Pattern Language for Programming Contest through Fight between Computer Players,″ 2nd Asian Conference on Pattern Languages of Programs, pp.1-18, October 2011.
(13)	坂本一憲, 東海政治, 村上裕子, 宮原里枝, 奥村有紀子, 秋山浩一 他 2 名, ″Web アプリケーション開発における画面仕様書およびテスト仕様書の自動生成手法と開発プロセス改善の提案,″ ソフトウェア品質シンポジウム 2011 論文集, pp.1-8, September 2011.
(14) ○	Kazunori Sakamoto, Takuto Wada, Hironori Washizaki, and Yoshiaki Fukazawa, ″テストカバレッジに基づくテストコードの再構成パターン,″ 1st Asian Conference on Pattern Languages of Programs, pp.11-15, March 2010.
(15)	塩浜龍志, 坂本一憲, 久保秋真, 鷲崎弘宜, 深澤良彰, ″アジャイル開発における適切なイテレーション期間のシミュレーションによる推定,″ ソフトウェアエンジニアリングシンポジウム 2011 論文集, pp.1-6, September 2011.
(16) ○	坂本一憲, 和田卓人, 鷲崎弘宜, 深澤良彰, ″テストカバレッジに基づく重複テストコードの検出ツール,″ ソフトウェアエンジニアリングシンポジウム 2010 論文集, pp.133-138, September 2010. （コンピュータサイエンス領域奨励賞）
(17)	坂本一憲, 鷲崎弘宜, 深澤良彰, ″テスティングフレームワークにおける問題と考察,″ 第 17 回 ソフトウェア工学の基礎ワークショップ論文集, pp.193-194, November 2010.

早稲田大学　博士（工学）　学位申請　研究業績書

種 類 別	題名、　発表・発行掲載誌名、　発表・発行年月、　連名者（申請者含む）
(18) ○	坂本一憲, 鷲崎弘宜, 深澤良彰, ″柔軟かつ複数プログラミング言語対応のテストカバレッジ測定フレームワーク,″ 第 8 回情報科学技術フォーラム論文集, pp.103-112, 2009. （船井ベストペーパー賞、船井ヤングリサーチャー賞）
2.講演 (研究会) (19)	坂本一憲, 海津智宏, 他 3 名, ″Web アプリケーションの動的部分に着目したグレーボックス統合テストの提案,″ 情報処理学会第 176 回 SE 研究発表会, pp.1-8, May 2012.
(20) ○	坂本一憲, 大橋昭, 太田大地, 鷲崎弘宜, 深澤良彰, ″UNICOEN: 複数プログラミング言語対応のソースコード処理フレームワーク,″ 情報処理学会 第 88 回プログラミング研究発表会, pp.46-46, March 2012.
(21) ○	Kazunori Sakamoto, Takuto Wada, Hironori Washizaki, and Yoshiaki Fukazawa, ″A Tool For Detecting Duplicated Test Code Based On Test Coverage to Assist TDD,″ 電子情報通信学会ソフトウェアサイエンス研究会, pp.41-46, June 2011.
(22)	坂本一憲, 鷲崎弘宜 他 1 名,″動的なコードの評価機構を備えた言語に対するテストカバレッジ測定ツール,″ 日本ソフトウェア科学会第 27 回大会, pp.153-158, September 2010.
(23) ○	坂本一憲, 大橋昭 他 10 名, ″AI プログラミングを通して参加する教育向けゲームシステムに適したソフトウェアパターン,″ ソフトウェアのパターンとアーキテクチャ・アジャイル開発, 情報処理学会ソフトウェア工学研究会, pp.7, August 2010.
(24) ○	坂本一憲 他 8 名, ″AI プログラミングを通して参加する教育向けゲームシステムに適したソフトウェアアーキテクチャ,″ ゲーム学会 第 3 回研究会, pp.10-15, March 2010.
(25) ○	坂本一憲, 鷲崎弘宜, 深澤良彰, ″共通の言語モデルを用いた複数プログラミング言語対応のテストカバレッジ測定フレームワーク,″ 第 7 回ディペンダブルシステムワークショップ, 日本ソフトウェア科学会, pp.167-170, July 2009.
(26)	下條清史, 坂本一憲, 鷲崎弘宜, 深澤良彰, ″プログラムの構造に着目した Fault-Localization とデバッグ支援,″ 信学技報, SS2011-73, pp.97-102, 2012.
3.著書 (1)	訳:小林健一, 吉野雅人, 太田大地, 坂本一憲, 小島 努, 原著:William C. Wake et al., ″リファクタリング Ruby 実践ワークブック,″ ピアソン桐原, November 2010.
4. その他 (ポスター) (1)	坂本一憲, ″AI プログラミングを通して参加する教育向けゲームシステムに適したソフトウェアアーキテクチャ,″ CESA デベロッパーカンファレンス 2010, August 2010.
(招待講演) (2)	坂本一憲, ″単体テストツールにおける複数言語対応化の事例とカバレッジ技術の最新動向,″ ガイオ プライベートセミナー2012 秋, November 2012.
(カリキュラム) (3)	芦田宏直, 芦澤昌彦, 鷲崎弘宜, 坂本一憲 他, ″オブジェクト指向教材開発「本格的なプログラマー教育のための４年制カリキュラム」,″ 文部科学省委託報告書, 2009.