Vol.2013-SE-179 No.16
2013/3/11

プログラミング演習支援のための細粒度履歴収集環境の開発

森一樹[†1]　田中昂文[†2]　橋浦弘明[†3]　櫨山淳雄[†2]　古宮誠一[†1]

高度なソフトウェア開発技能を持った人材を育成するためには，基礎となるプログラミング技術を早期にかつ確実に習得させる必要がある．プログラミングの教育は演習を中心に行われており，演習による知識の定着の度合いを評価するために，成果をレポートとして提出させる場合が多い．レポートは演習の最終結果を確認することはできるが，学習者がソフトウェアを完成させていく過程や，その途中でどのような問題に直面し，どのように対処したかを教授者が知ることは難しい．本研究では Eclipse と連携して，学習者のプログラミングの細粒度履歴をリアルタイムに収集し，ネットワークで接続されているサーバに蓄積することにより，ソフトウェア開発演習を支援する環境を構築した．

Development of an Environment for Gleaning History Information of Programming on a Fine Granularity to Help with Programming Exercise Lesson

KAZUKI MORI[†1]　TAKAFUMI TANAKA[†2]　HIROAKI HASHIURA[†3]
ATSUO HAZEYAMA[†2]　SEIICHI KOMIYA[†1]

1. はじめに

高度なソフトウェア開発技能を持った人材を育成するためには，基礎となる要素技術を早期かつ確実に習得させる必要がある．要素技術のなかでプログラミングは大きな割合を占めているが，プログラミング演習では，その機能性の実現についての教育が重要視され，モジュール分割の適切さやコードの可読性など，品質に対する評価や指導が後回しになっているという現状がある．

現在多くの教育機関で行われているプログラミング演習では，授業中に与えられた到達目標を達成しているかどうかを確認・評価するために，学習者が授業の合間や終了後に提出したレポート（完成したファイル）を活用している．レポートの形式を用いると，教授者からのフィードバック回数はレポート提出後（次回授業開始時など）の 1 回のみであり，授業中に指摘してすぐに改善を行う，といった行動がとりにくい．また，レポートの出来は学習者個人によって異なるが，完成度が低いものに対しては，なぜ完成に至らなかったのか，どこに手間取ったのか，といった課題の作成過程が分からないため，理解を促進するための補助資料の作成が難しい，といった問題点がある．

このような問題を解決するために，プログラミングの初期段階からソフトウェア工学的観点に則り，段階的かつ継続的にソフトウェア品質についての教育を支援する環境を提案する．

本研究では，プログラムの履歴収集ツールをプログラミング演習に導入し，演習中に送信された履歴を自動的に解析し，品質についての評価を学習者にフィードバックすることにより，プログラムの品質を学習の早い段階から教授者や学習者に意識させ，自発的な改善を促す．また，得られたデータを分析することにより，学習者が陥りやすい典型的な問題を検出する方法について述べる．

本稿における構成を以下に述べる．2 章では，ソフトウェア工学教育の現状と成果物評価の問題点について述べる．3 章では，関連研究との比較を述べる．4 章では，細粒度履歴を取得し，分析するためのアプローチについて述べる．5 章では，本研究で提案するプラグインのアーキテクチャについて述べる．6 章では，プラグインの評価について述べる．7 章では，本研究のまとめと今後の課題について述べる．

2. ソフトウェア工学教育の現状と成果物評価の問題点

1 章で述べたような実践的なエンジニアを育てるための一つのアプローチが PBL（Project-Based Learning）と呼ばれる方法である．実社会ではプロジェクトチームを組織してソフトウェアを開発するという形態が一般的である．そこで，演習授業などにおいて幾つかのグループに分かれて行うソフトウェア開発の演習を小さなプロジェクトに見立て，その中で実践的なソフトウェア開発を学習者らに疑似体験させるのが PBL の特徴である．このような方法は以前からその有効性が認められており，著者らも同様な取り組

†1　芝浦工業大学大学院理工学研究科
Shibaura Institute of Technology
†2　東京学芸大学
Tokyo Gakugei University
†3　東洋大学
Toyo University

みを行っている．

しかしながら，ソフトウェア工学教育は PBL を行うことさえすれば，十分なトレーニングができるというわけではない. 松浦[1]は PBL を実施するにあたっては以下の点を踏まえる必要があると指摘している．

- （PBLの主な実施時期である）学部3年後期までにある程度の規模のソフトウェア開発が行える前提知識を効率よく教授する必要がある
- 講義で得た知識を活用しながら主体的に問題解決を行う力を養うが，グループとして行動するため個人の成績評価が難しい

実際にある程度の規模や複雑度をもつような課題で PBL を行ってみると，松浦が指摘しているように，PBL 開始前に習得しておくべき前提知識のうち，特にコードの品質に対する理解が不十分であるために，質の低いコードを生産してしまい，そのことが演習完遂の障害となっている例が散見される．また，PBL はグループで演習を行うため，多人数でコードを共有しているため，コードの品質の問題を指摘しても，ソフトウェア開発のための知識やスキルが学習者個人のレベルで習得されることになるとは限らないという問題もある．

このような問題は,「コードの品質をいつ教えるか」という問題と直結している．ソフトウェアを初めて学ぶ学習者は，まず初歩的なプログラミングから学習を開始し，座学で得た知識を活用しながら，少しずつ難しい問題に取り組むこととなる．

ソフトウェア品質特性については ISO/IEC9126-1[2]において 6 種類に大別されている．演習の評価においてはこれらの品質を偏りなく評価することが，ソフトウェア工学教育上，重要である．竹田ら[3]の研究のように，PBL において仕様変更を行うことにより品質の問題を間接的に学習者に意識させるようにする方法もあるが, 著者らは PBL を行う前の基礎的な演習の段階から機能性や使用性という観点からプログラムの品質について教えることや，悪所を指摘する必要があると考えている．

プログラムの品質の評価が機能性に偏ってしまう原因として考えられるのは，機能性は実際にソフトウェアを動作させてみればそれらを評価することが比較的容易な特性であることや，演習課題であっても，ソフトウェアは特定の目的のために作られており，目的を満たさないプログラムは，他の品質特性が如何に優れていたとしても実用とすることができないことが挙げられる．加えて，プログラミングの初歩の段階では，目的を達成するための技術の習得に重点が置かれており，目的を明確化するために，習得すべき技法に対して問題が単純化されていることが多いことも，このような傾向に拍車をかけている．

一方，評価において保守性や効率性などに対する評価は後回しになりがちである．これらの品質に関する問題は，規模や複雑さが大きくなるにつれて弊害が顕在していく性質を持っているが，前述のような状況においては，学習者は品質に関する問題を自分自身で認識しにくく，かつ，教授者も前述の問題構造からこのような品質の問題を教えにくい状況にある．

また，これらの品質特性を評価するためには，単にソフトウェアを動作させるだけでなく，実際に作成されたソースコードの内容を詳細に調査する必要がある．学習者が抱えるコードの品質問題は個人個人で異なっており，このような作業を多人数の学習者に対して指導を行うためには膨大なリソースが必要となることも教授者にとっては大きな障害となっている．

井上ら[4]は，オブジェクト指向設計過程において発生する，典型的な誤りを学習者に修正させるためには，修正方法を示さずに，誤りが発生している箇所を示すだけで十分であると述べている．従って，本研究では演習で送信された履歴を自動的に解析し，解析結果を学習者にフィードバックすると共に，学習者全体で共有する環境の導入を提案する．

学習履歴を用いたフィードバックを可能とする環境の導入により，学習者に自主的に品質の高いソフトウェアの作成を促すと共に，自力で修正が難しいものについては，教授者が個別に指導することができるようになる．学習者，教授者それぞれに期待される効果について下記に示す．

学習者への期待される効果
- 自分の作成したソフトウェアの品質特性（どこが悪いのか，どこが良いのか）を認識することができる

教授者への期待される効果
- 躓いている学習者の発見につながる
- 難易度の調整やフォローに役立てることができる

3. 履歴を用いたプログラミング学習支援に関する関連研究

これまでにもプログラミングの過程に着目した研究の提案が行われている．谷川ら[5]は学習者の理解度を把握するためプログラミング演習中に発生する学習者の試行錯誤に着目し，スパイ関数と演習経過記録ツールを用いることでプログラム実行時の関数の呼び出しの順序を記録することを提案している．スパイ関数はリンカによって学習者のプログラムに埋め込まれ，学習者が作成したプログラムを実行することによって, 実行時に呼び出し履歴が記録され，演習経過記録ツールを通して可視化される．谷川らはツールによって得られた履歴から，学習者の試行錯誤の内容を

考察し，学習の過程と誤りのパターンを導出している．しかしながら，スパイ関数を用いて関数の呼び出し順序を得るためには，少なくとも学習者がプログラムを起動できる段階まで課題を完成させた上で学習者が試行錯誤を行うことが前提となっており，プログラムを起動させる状態に到らない場合の支援は期待できない．

長谷川らはプログラミング演習において，演習時間内に学習者に助言を与えることができる統合リアルタイム授業支援システム IDISS[6]を開発している．IDISS は学習者に課題を提示し，学習者が課題を解いてシステムに提出を行うと，プログラムの自動評価を行い，プログラムの機能性や非機能性についての評価を行って，結果を学習者に提示する．また，システムは評価をシステム内に保存し，教授者に提示する．このようなシステムは学習者が多数存在する演習授業には有効なアプローチであると考えられるが，学習履歴の保存タイミングについては学習者の提出回数に依存するため，学習者間での履歴の取得の密度に差が生じるという問題が残っている．これらの問題については統合開発環境（IDE）を用いた自動的な履歴取得が有効であり，すでにいくつかの提案も行われている．

大森と丸山[7]はプログラムを理解するためには，ある時点の特定のソースコードだけでなく，そのソースコードがどのように変更されてきたのかを知る必要があり，そのためには従来のバージョン単位の履歴では不十分で，ユーザーの操作の単位で記録を取ることの有効性を指摘している．さらに，統合開発環境（IDE）向けにユーザーのソースコード編集操作履歴を記録するプラグイン Operation Recorder を開発して，ソースコード版管理システムよりも細粒度の開発履歴の取得と操作履歴の再現ができることを示している．しかしながら，Operation Recorder はプログラミング演習を意図したツールではないため，得られた履歴をどのように活用するかについては応用性を述べるに留まっている．

Hattori らが開発した Syde[8]はチームによるプログラム共同開発を対象として，ファイルよりも細粒度な履歴をリアルタイムに収集し，分析することにより，チームにおいて現在誰がどの部分の変更を行っているか，変更箇所に競合が発生しているかなど，分析結果の通知をリアルタイムに行う．通知はチーム全員の IDE 上に即座に反映されるようになっており，その際に学習者の関与は一切必要ない．ただし，Syde ではソフトウェア品質の向上を支援するような環境は構築できない．

これらの関連研究では，「プログラムが動作して初めて履歴が取れる」「履歴の保存タイミングが学習者に依存する」「プログラミング演習への適用を考えていない」などの問題点が存在し，学習者が躓いている部分を細かく知り，躓きを解消するためのフィードバックはできない．本研究では，プログラミング演習における学習者の細粒度の開発履歴をリアルタイムに収集し，分析することで，学習者への迅速なフィードバックができるような環境を整える．

4. 本研究のアプローチ

本研究では，学習者がどのような過程を経て課題を完成させるに至ったのかという，学習者の行動の履歴に着目する．プログラミング演習における学習者の行動の履歴は，テキストエディタにおける編集操作とソースコードのコンパイル，プログラムの実行の履歴である．これらの履歴を収集することで，プログラミング演習における学習者の過程を明らかにするための環境を構築する．

なお，履歴を収集するために，学習者が何らかの手順を踏んで設定を行う，ということはあってはならない．学習者はストレスを感じることなく，リアルタイムで履歴を送信する必要がある．このため，本研究では IDE の Eclipse に履歴収集用のプラグインを開発・導入し，学習者が Eclipse を通して入力した履歴をサーバに逐次送信するという手法をとる．Eclipse を用いた Java 言語のプログラミング演習は広く行われており，将来的には演習授業への本システムの導入を行いたいと考えている．

本研究ではプログラミングの過程を詳細に把握するために行やファイル単位ではなく，ユーザーの行動単位での履歴の記録（以下，細粒度履歴と呼ぶ）を行う．細粒度履歴とは，IDE における下記の操作の流れとして表わされる．

A) エディタへの文字の入力
B) エディタへの文字の削除
C) エディタへの文字の変更
D) ファイルの保存
E) プログラムの実行

この細粒度履歴情報は人間が扱うには詳細すぎる情報なので，多数の学習者の細粒度履歴を全て閲覧して，学習の過程を理解することは難しい．従って，細粒度情報をプログラミング過程がわかりやすいように加工する．

教授者へのフィードバック（学習者がどのように課題を完成するに至ったか）には，「クラス・メソッドを作成するのにどれほど時間が掛かったのか」「クラス・メソッドを作成するためにどれほどの手数（文字入力・削除やコピー&ペースト）を掛けたのか」「どういった順番でファイルを編集していったのか」といった履歴に加工した上で提示すればよい．また，エディタへの文字の編集から取得できる履歴のうち，1 文字ずつの編集履歴では理解がしづらく，学習者が一時に纏めて行った編集の履歴に加工した方が理解しやすい．よって，エディタから取得した細粒度履歴をそのまま使うのではなく，上記のような形式で履歴を統合・

加工した後，教授者へのフィードバックを行う．また，「どの学習者がどの履歴を送信したのか」という判別が出来るようにする．これにより，授業中に学習者が送信した履歴を見れば，各学習者の進捗状況が分かり，学習者への指導もしやすくなる．

学習者へのフィードバックには，演習を行っている際に，ソースコードのどの部分の品質が悪いのか，を指摘する必要がある．これについては，教授者がシステムからフィードバックを受け，学習者の進捗状況を把握した上で，品質が悪い箇所について口頭で伝えればよい．また，システムを通して学習者のエディタに指摘箇所を表示させることでフィードバックを行うこともできる．この際には，学習者が品質を守るために順守すべきルールをシステムに入力することで，そのルールに沿わない部分のコードに警告を発する．

5. 細粒度履歴収集プラグイン

ここで，本研究で開発した細粒度履歴収集を可能とするEclipse プラグインである Verdure について述べる．Verdure は Eclipse の Java 編集用のテキストエディタプラグイン（Java エディタ）をベースとしており，Java 言語の演習における履歴収集を可能としている（図 1）．

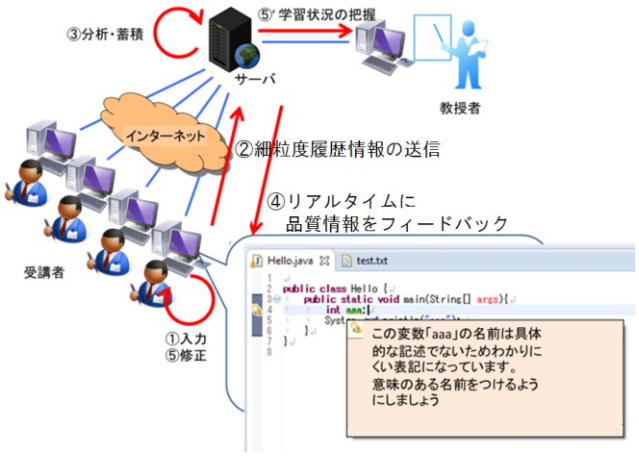

図 1：Verdure の利用イメージ

5.1 Verdure の構成

Verdure は，以下の 3 つのモジュールと，1 つのサーバから構成される（図 2）．

(1) VerdureEditor：学習者の Java 言語による開発履歴を収集し，フィードバックの表示を行うテキストエディタ
(2) Verdure.io：収集した細粒度履歴を分析し，送信するプラグイン
(3) サーバ：Verdure.io から送信された履歴を保持し，学習者，教授者へのフィードバックを行うサーバ
(4) VerdureLogViewer：収集した履歴に基づいて，学習者の開発過程を再生するツール

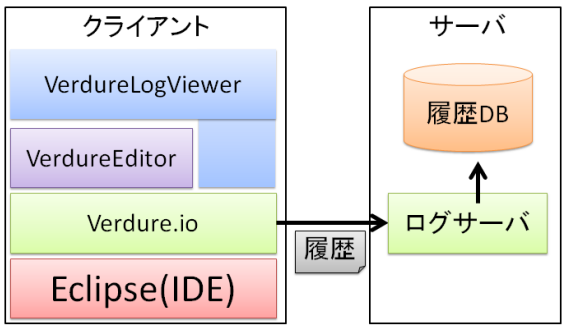

図 2：Verdure の構成

5.2 VerdureEditor

VerdureEditor は Eclipse の Java エディタを拡張し，次の情報の収集を行う．

1. どの学習者が生成した履歴か
2. ファイルが読み込まれたときのファイルの初期状態
3. エディタに入力された文字，入力した時刻
4. エディタから削除された文字，削除した時刻
5. エディタ上で行われたアンドゥ（文字の切り取り，貼り付け，もとに戻す：Ctrl＋Z，やり直す：Ctrl＋Y，フィールド・メソッド・クラス名のリファクタリング）
6. Eclipse が終了したときのファイルの最終状態

また，教授者から品質上守るべきルールが追加されたとき，そのルールに沿わないコードが存在する場合には，図 3 のようにエディタ上に警告を発生させる．

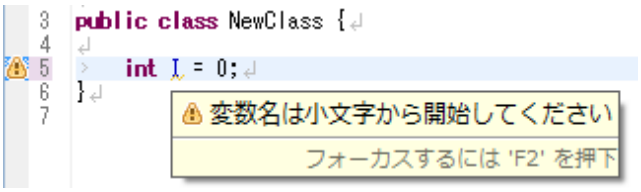

図 3：教授者からのフィードバック

5.3 Verdure.io

Verdure.io は VerdureEditor で作成された履歴の分析を行う．VerdureEditor では，文字の入力，削除の履歴は 1 文字ごとに取得される．たとえば，”String” という文字列を入力したときには，S, t, r, i, n, g という 6 個の履歴が収集される．教授者が必要とするのは 1 文字ごとの履歴よりも，ある程度まとまった履歴である．つまり，”String”と入力されたならば，「”String”という文字列が入力された」という一つの履歴があった方が，理解がしやすくなる．

文字の削除についても同様で, ”text”という 4 文字を連続して削除した場合には, VerdureEditor では t, e, x, t が 1 文字ずつ削除されたという 4 個の履歴が収集される. このとき, Verdure.io では,「”text”という 4 文字が削除された」という 1 つの履歴に統合する処理を行う.

また，学習者がカーソルを移動することなく，連続して文字を編集し続けている場合に関しては，1 つの履歴として統合する．たとえば，”int a”と入力した直後に，カーソルを移動することなく”a”を削除し，”num”を入力したような場合には，最終的に「”int num”という文字列が入力された」という 1 つの履歴に統合する．

統合された履歴は，リアルタイムにサーバに送られるほか，学習者が Eclipse を終了したときにも送信される．現在行われているレポート形式の課題提出方法では，授業の終わりに必ずファイルサーバ等にレポートをアップロードしなければならず，そのために授業内で 5 分程度の時間が浪費されてしまう．しかし，Eclipse 終了時（演習終了時）に自動的に完成ファイルを送信することにより，その手間を省くことができ，学習者はより演習に専念することができる．

5.4 サーバ

サーバでは, Verdure.io から送信された履歴を学習者ごとに管理する．教授者から履歴の取得命令があると，指定された学習者の履歴情報を VerdureLogViewer に送信する．

また，教授者から学習者へのフィードバック（コーディングの指摘など）がある場合には，サーバから学習者の VerdureEditor に指摘情報を送信する．

5.5 VerdureLogViewer

VerdureLogViewer は，教授者が学習者の開発過程を再生するための，Eclipse プラグインである．課題の出来が悪い学習者がどこに躓いているのか等を知るために，その履歴を再生することで，学習者へのフィードバックのための分析を可能にするプラグインである．

このプラグインでは，Eclipse 上から学習者，履歴の存在するファイルを選択し, コマンドを実行することによって, 履歴の 1 ステップごとに再生することができる（図 4）.

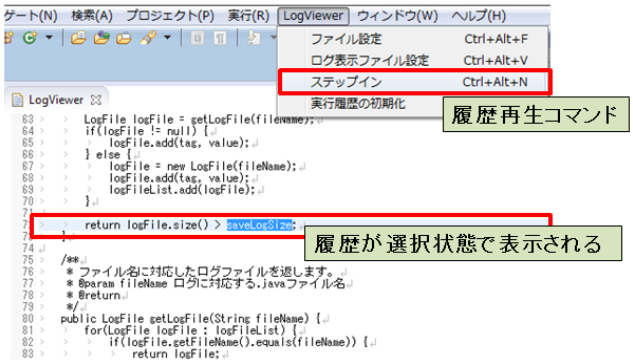

図 4：VerdureLogViewer

6. 評価

本章では, Verdure を実際に使用したときに取得した履歴を紹介する．履歴を取得するにあたり，被験者 1 名に「コンソールから半径（整数値）を読み取り，円の面積を表示するプログラム」の作成をしてもらった．取得したログの一部が以下の表 1 である．

表 1：取得したログの一部

#	fileName	tag	text	offset	date
1	Main	Insert	package main; public class Main { }	0	14:18:25
2	Main	Insert	public static void main()	36	14:18:26
3	Main	Insert	String args[]	66	14:18:31
4	Main	Insert	{ }	80	14:18:34
5	Main	Insert	System.out.println()	86	14:18:38
6	Main	Insert	“”	97	14:18:41
7	Main	Insert	半径を入力してください。	106	14:18:44
8	Main	Insert	;	120	14:18:48
9	Main	Insert	public static int input() { }	125	14:18:57
10	Main	Insert	InputStreamReader isr = new InputStreamReader(System.in); BufferedReader br = new BufferedReader(isr); String buf = br.readLine();	160	14:19:13
11	Main	Insert	import java.io.InputStreamReader; import java.io.BufferedReader;	15	14:19:20
12	Main	Delete	buf	384	14:19:43
13	Main	Insert	input	384	14:19:16
…	…	…	…	…	…

表 1 では，履歴についてそれぞれ以下の情報を表している．

(a) fileName：編集を行ったファイル名
(b) tag：履歴の種類（Insert：書込，Delete：削除，など）
(c) text：挿入された，文字列または削除された文字列
(d) offset：編集が行われた開始点
(e) date：履歴を取得した時刻

Verdure では履歴を取得した際の時刻も記録されるため，どの部分を編集するのにどれほどの時間をかけたのか（どれほど考えるのに時間をかけたのか），が判る．また，履歴は入力・削除のまとまりとして取得されるため，履歴を 1 ステップずつ再生したときに，学習者がどの部分を編集したのかが判別しやすい．

図 5 は，取得した履歴を 1 つずつ再生していき，履歴#11 を再生したときの様子である．図 5 の選択部分がエディタへの変更点を表している．#11 では，かなり長い文字列が一度に入力（貼り付け）されていると判る．履歴を見ることで，学習者同士の成果物の受け渡しなどの不正もすぐに判別できる．また，入力と削除が繰り返されているような部分を発見すれば，そこは学習者が試行錯誤を繰り返していた部分とも判る．

このように，Verdure を用いれば，リアルタイムに学習者が躓いている部分を発見することができ，フィードバックをしやすい・受けやすい環境を作ることができたと言える．

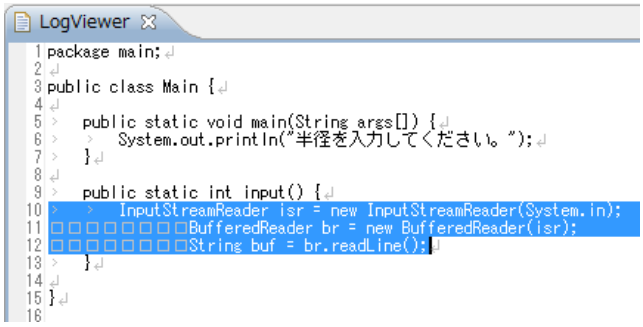

図 5：履歴#11 を再生した様子

7. まとめと今後の課題

本研究では，プログラミング演習において，モジュール分割の適切さやコードの可読性など，品質に対する評価や指導が後回しになっているという現状に対して，学習者へのフィードバックを行うための基盤として，Eclipse プラグインによって細粒度の開発履歴を取得するというアプローチを行った．リアルタイムに開発履歴を収集し，分析することで，学習者にとっては自分のコードの良い点，悪い点のフィードバックを受けやすくなるという利点がある．また，教授者にとっては学習者がどんなところで躓いているのかが判り，難易度の調整に役立てることができる，という利点がある．

ただ，現在このツールでは，収集した学習者全員の履歴を自動で分析することや，その分析結果をもとに品質を向上させるための案を提示することができない．今後は履歴の詳細な分析方法や，その活用について議論を進める必要がある．

参考文献

1) 松浦 佐江子, "実践的ソフトウェア開発実習によるソフトウェア工学教育," 情処論, Vol.48, No.8, pp.2578-2595, 2007.
2) “ISO/IEC 9126-1:2001 Software engineering - Product quality - Part 1: Quality model,” International Organization for Standardization, 2001.
3) 竹田 尚彦, 大岩 元, "プログラム開発体験に基づくソフトウェア技術者育成カリキュラム," 情処論, Vol.33, No.7, pp.944-954, Jul. 1992.
4) Yuichi Inoue, Hiroaki Hashiura, Seiichi Komiya, "A Method for Detecting and Avoiding Many-to-Many Relationship in Class Design of Software Object-Oriented Development," Communications of SIWN(CoSIWN), Vol.6, pp.158-165, Apr. 2009.
5) 谷川紘平, ディン ドン フォン, 原田 史子, 島川 博光, "C 言語呼び出しの記録を 用いた演習過程での習得項目の把握 ," 信学論 D, Vol.J95-D, No.12, pp.2079-2089, 2012.
6) 長谷川 伸, 松田 承一, 高野 辰之, 宮川 治, "プログラミング入門教育を対象と したリアルタイム授業支援システム," 情処論, Vol.52, No.12, pp.3135-3149, 2011.
7) 大森 隆行, 丸山 勝久, "開発者による編集操作に基づくソースコード変更抽出," 情処論, Vol.49, No.7, pp.2349-2359, 2008.
8) L. Hattori and M. Lanza, Syde: A Tool for Collaborative Software Development, Proc. 32th International Conference on Software Engineering (ICSE2010), pp. 235-238, ACM Press, 2010.