Vol.2015-SE-187 No.13
2015/3/12

ソフトウェア設計支援ツール Perseus における模範解答一致判定機能の開発と評価

柴田祐貴[†1]　　掛下哲郎[†1]

近年のソフトウェアの大規模化に伴い系統的なソフトウェア設計を行うことは重要な課題となっている。そこで、我々は系統的なソフトウェア設計を教育するための支援ツール Perseus を開発している。本論文では Perseus に模範解答との一致判定機能を実装して評価を行う。Perseus は DOM を用いた木構造でソフトウェア設計を表現している。そのため、模範解答との一致判定ではツリーマッチングを利用して対応するノードを決定する。また、ノード間の一致判定を行う際にはレーベンシュタイン距離を用いる。学生の答案と模範解答の記述に表記の揺れがあったときに正しい判定を行えない場合があるため、表記の揺れに対応できるように模範解答に別解を付加できるようにした。模範解答の木構造は Perseus の設計木構造と構成が異なるため、我々は模範解答編集ツール Pras.Edit を Perseus とは別に開発した。Pras.Edit は模範解答の編集や暗号化・復号化、Perseus で作成した木構造を模範解答に変換する機能を提供する。さらに、模範解答との一致判定機能の評価実験を実施し、設計ミスの検出状況について調査を行った。一致判定機能を用いることで、手作業で添削した場合と比較して 3.3 倍のミスを効率よく指摘できたが、添削ミスが全体の 21.8% 発生した。しかし、別解付加機能を用いることで添削ミスを 14.3%に減らすことができた。

Development and Evaluation of Matching Function with Model Answer for Software Design Support Tool Perseus

YUKI SHIBATA[†1]　TETSURO KAKESHITA[†1]

Systematic software design is an important issue in recent years according to the increase of software size. We develop a software tool Perseus for systematic software design education. In this paper, we develop and evaluate the matching function for Perseus between the model answer and student's answers. Perseus represents software design as a tree structure using DOM. Therefore, the matching function decides corresponding nodes using tree matching. Matching of nodes utilizes the Levenshtein distance. We also develop functions to extend the model answer to integrate alternate answers. Moreover we develop a model answer editor Pras.Edit. Pras.Edit provides functions to edit, encrypt and decrypt model answer in addition to the conversion function from a tree structure created using Perseus. We perform an evaluation of the matching function utilizing 20 student answers. The number of mistakes detected by the matching function is 3.3 times larger than that of the manual scoring. 21.8% of the detected mistakes are not correct. However, the number of incorrect matching is reduced to 14.3% by using alternate answers.

1. はじめに

近年、ソフトウェアの大規模化・複雑化が進んでおり、系統的なソフトウェアの作成は重要な課題となっている。ソフトウェアの標準的な開発プロセスを定義した共通フレーム 2013[1]においても、ソフトウェアの作成は要件定義、方式設計、詳細設計、構築、テストの順で行うとされている。方式設計、詳細設計の段階で良い設計を行うことにより再利用性やモジュール間の独立性、ソフトウェア保守性等、様々なソフトウェア品質を向上できる。しかし、ソフトウェア設計には要求分析や仕様作成、コーディングなどの様々な工程の知識が求められる。また、演習には多くの時間がかかることもあり、大学の授業としてうまくまとめて教えるのは困難である[2]。そこで、我々は系統的なソフトウェア設計を効率的に指導するためにソフトウェア設計教育支援ツール Perseusを開発してきた[3]。

Perseus はソフトウェア設計を系統的に教育することを目的に開発されており、教員は Perseus を用いて多数の学生を指導している。教員は学生の作成したソフトウェア設計を個別に点検するため手間がかかる。また、提出されたソフトウェア設計には同じようなミスが多く、中には未完成の設計を提出する学生もいる。そこで、学生が作成した設計結果の点検を自動化して添削の効率化を図るとともに添削品質の向上を目指す。これを実現するために、本論文では Perseus に模範解答との一致判定機能を追加する。

一致判定機能は、学生の設計結果と教員の作成した模範解答を比較し、記述が一致しない場合には該当箇所にコメントを付加する。一致判定にはレーベンシュタイン距離を用いており、閾値を変更することで一致範囲を制御できる。

模範解答一致判定機能で使用する模範解答は、Perseus で作成する設計とは必要となるデータが異なっている。そのため、本論文では模範解答一致判定機能と平行して模範解答編集ツール Pras.Edit（Perseus Right Answer Editor）の開発を行う。Pras.Edit は作成した模範解答の暗号化・復号化

†1 佐賀大学
Saga University

機能や、Perseus で作成された設計結果を模範解答に変換する機能を提供する。

本論文では、模範解答一致判定機能の開発と評価、及び Pras.Edit の開発を行ったので、これについて報告する。第 2 節では、ソフトウェア設計支援ツール Perseus の機能について説明する。3 節では、Perseus に実装した模範解答一致判定機能の内容やレーベンシュタイン距離、マッチング方法について述べる。4 節では、模範解答のデータ構成の変更と模範解答を作成するために開発した模範解答編集ツール Pras.Edit の設計と機能について説明する。5 節では、模範解答一致判定機能の評価実験について、その内容及び分析結果を示す。

2. ソフトウェア設計支援ツール Perseus

2.1 概要

Perseus はソフトウェア設計学習者を対象として作成されたソフトウェア設計支援ツールである。設計と実装を分離することにより開発方法論やプログラミング言語等に依存することなく系統的なソフトウェア設計を行うことができる。現在、佐賀大学・知能情報システム学科で開講している専門必修科目「ソフトウェア工学」においても、モジュール設計演習および Jacson 法による構造化プログラミング演習を行う際に Perseus を活用している。

Perseus は DOM を利用した木構造を用いてソフトウェア設計を表現している。各設計要素は木構造のノードや部分木に対応しており、図 1 の UI を用いてこれらを組み合わせることによりソフトウェア設計を表現する。これにより、モジュール設計、ルーチン設計、データ構造設計、アルゴリズム設計など、さまざまなレベルのソフトウェア設計に対応している。また、Perseus ではログ機能等に対応するために各ノードには設計情報を保持する comment、ノードを判別する node-id、コメントを保持する review-comments の 3 つのデータを持つ。

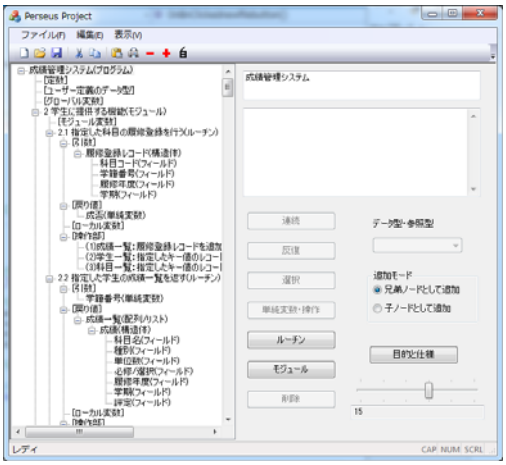


図 1 Perseus のユーザインターフェース

2.2 機能

Perseus は主要な機能である設計木構造の編集機能とソフトウェア設計を支援する機能を提供する。

設計木構造機の編集

学習者は Perseus を用いて操作対象ノードを選択し、そのノードに対して操作を行うことで木構造の設計を行う。ノードに対して行える操作は、設計テキストの変更、設計部分木の追加、ノードの削除・コピー・切り取り・貼り付け、ノードの展開・縮約である。設計部分木の追加は設計が矛盾しないようノードの種類に応じて追加できる部分木の種類が制限されている。同様に、ノードの削除・コピー・切り取り・貼り付け操作においても操作できるノードに制約があり、設計の一貫性が保たれる。ノードの展開・縮約は各ノードに個別で操作できるほか、木構造の全てのノードに対して一括で操作が行える。設計した木構造は保存、読み込みを行うことができる。拡張子は prsx としている。

ログ機能

Perseus は学習者の操作履歴をログに記録する。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。ログファイルは学習者が設計木構造を保存した際に CSV ファイルとして出力される。また、ログファイルを読み込み学習者の操作を再現することが可能であり、ソフトウェア設計の構築過程を調べることもできる。

レビューコメント付加機能

教員は設計木構造のノードに対してコメントの付加、削除を行うことができる。付加するコメントは手動で入力する以外にもデフォルトで用意されたものがある。入力されたコメントを保持することで同一のコメントを複数のノードに付加できる。コメントは CSV ファイルとして保存、読み込むことができる。

PDF 出力機能

現在の設計木構造の内容を PDF として出力する。出力する情報はノードの設計情報のほか、ソースコード、仕様と目的、レビューコメントがある。

3. 模範解答との一致判定機能

3.1 概要

Perseus はソフトウェア設計を教育する際に使用しており、教員は複数の学生に対して指導を行っている。指導に際して教員は学生が作成した設計木構造の添削を行うが、多数の設計木構造の添削を行うのは手間がかかる。既存の添削支援機能としてレビューコメント付加機能が提供されているが、この機能では設計木構造に対して手動でコメントを付加する必要がある。また、作成された設計木構造には同じようなミスが多く、中には未完成のものが提出されていることもある。これらを一つ一つレビューしていては手間がかかる。このレビューにかかる手間を削減するため

に Perseus に模範解答との一致判定機能を追加する[4]。

本機能を実行すると設計木構造と模範解答の比較を行い、結果に応じて対応したノードにレビューコメントが付加される。実行前にノードに付加されていたコメントは全て削除される。これにより、学生のソフトウェア設計に対する添削を自動化し添削の効率化を図ることができる。コメントは Perseus 上で該当ノードを選択することで表示される。また、既に模範解答が読み込まれている状態で学生の設計木構造を新しく読み込む、あるいはスライダーで閾値を変更した場合は自動で模範解答一致判定機能が実行されるようになっている。

3.2 レーベンシュタイン距離

模範解答との一致判定機能では学習者の設計木構造と模範解答の設計木構造の比較を行う。木構造同士の比較を行う際には、マッチングを行いそれぞれの木構造におけるツリー及びノードの対応付けを行う必要がある。本機能では、文字列の類似度を表すレーベンシュタイン距離を用いてノード同士の対応付けを行う[5]。

木構造の比較を行うに際し、定量的な値を用いたツリー及びノードマッチングを行う必要がある。そこで、文字列の類似度を定量的に表す値として、一方の文字列をもう一方の文字列に編集する際にかかる操作数を用いる。文字列の編集には様々な手順があるが、その中で最短手順での操作数を表すのがレーベンシュタイン距離である。編集操作には文字の挿入、削除、置換の 3 種類の操作を用いる。挿入、削除の操作数は 1 とし、置換は「削除してから挿入」を行っていると考えられるため操作数は 2 とする。

レーベンシュタイン距離は外部ライブラリである「dtl」[6]を用いて計算を行う。dtl ライブラリではレーベンシュタイン距離を求める際に iterator を用いていが、Visual Studio が提供するクラスである CString は iterator を持たない。そのため、全角文字と半角文字をどちらも 1 文字として扱えるようにするために TCHAR を要素とした Vector 配列を用いて比較することでレーベンシュタイン距離を計算している。

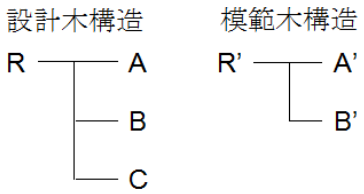
3.3 ノードマッチング

模範解答一致判定機能では、設計木構造と模範木構造のそれぞれのルートノードから末端のノードまで再帰的にマッチングを行いノード同士の対応付け及び設計ミスのチェックを行う。なお、ルーチンノードの子ノードは引数、戻り値、ローカル変数、操作部の 4 つの固定ノードのみで構成されるためマッチング処理を行わず直接対応付けしている。以下で、マッチングの処理について説明する。

(1)類似度表の計算

ノード同士の対応付けでは、設計木構造と模範木構造の対応するノードの子ノード群に対して再帰的にマッチングを行っていく必要がある。そこで、ノード同士の対応付けには行を設計木構造の各子ノード、列を模範木構造側の各子ノードと対応させた類似度表を使用する。

類似度表の各要素は各ノード同士のレーベンシュタイン距離である。なお、ルーチンノード部のマッチングの際には引数と戻り値の各ノードについて対応付けを行い、対応付けられたノード同士のレーベンシュタイン距離の合計値を類似度表の要素に加算する。なお、ルーチンノード同士のレーベンシュタイン距離が 0 であった場合は対応するノードであることが分かっているので加算しない。このとき、引数と戻り値のそれぞれの類似度表には各ノードの全ての子ノードを行及び列に対応させる。また、類似度表を作成する際にノード数が一致しない場合には、不足している方に文字列が空の要素を追加する。これは類似度表の行数と列数を一致させるためである。



	A'	B'	ϕ
A	2	9	8
B	8	4	9
C	6	8	7

図 2. 類似度表の作成

(2)貪欲法を用いたノードの対応付け

作成した類似度表を利用して最適なノード同士の組み合わせを決定する。決定するためのアルゴリズムとして、レーベンシュタイン距離の合計値が最小となるようにグリーディ法を用いる。表の全ての要素からレーベンシュタイン距離が最も小さいものを選択済み要素とし、対応する行と列を選択済み行列とする。表の残りの要素に対して同様の処理を繰り返し全ての行と列が選択済み状態になったら処理を終了する。この時、レーベンシュタイン距離が最小のものが同じ行あるいは列に複数存在した場合は、それぞれの要素の場合で計算を行い結果が最小となったものを選択する。こうすることで、二つの対応するノードの子ノード間でレーベンシュタイン距離が最も近いノード同士を対応付けすることでノードマッチングを行うことができる。

(3)マッチング結果の判定

グリーディ法を用いたマッチングでは同階層の全てのノードに対応付けが行われるため、空ノードと対応づけされたノードや類似度の低い対応付けが含まれる場合がある。これに該当するノードは設計ミスとしてコメントの付加対象になる。そこで、妥当なノードとのマッチングか余分なノードとのマッチングかの判定を行う必要がある。

マッチング結果の判定は対応したノード間のレーベンシュタイン距離とユーザーの指定した閾値を比較することで決定する。レーベンシュタイン距離が閾値より小さければ妥当なノードとのマッチングとし、大きければ余分なノードとのマッチングであると判定している。なお、空のノードとのマッチングについては閾値に依存せず余分なノードとのマッチングとする。

(4)コメントの付加と再帰処理の判定

余分なノードとのマッチングと判定されたノードは設計ミスと判断できる。それらのノードに対しては XPath 式を用いてコメントを付加する。

・設計木構造に余分なノードと対応付けされたものがある

学生が作成した設計木構造のノードのうち、模範木構造の空ノードや類似度の低いノードと対応付けされたものに対しては、「対応ノードが模範解答にない」旨のコメントを付加する。

・模範木構造に余分なノードと対応付けされたものがある

学生が作成した設計木構造のノードが模範木構造のノードと対応しない場合、当該ノードの親ノードに対して「模範解答に含まれている要素が記述されていない」旨のコメントを付加する。

次に、妥当なノードとの対応付けがされたノードがデータ構造部およびアルゴリズム部に該当するノードであった場合以下のチェックを行う。

・対応するノードの種類が異なっている

データ構造部であれば単純要素、配列/リスト、構造体、共用体の 4 つ、アルゴリズムであれば if/else 選択、for/while 反復、switch 選択、単純操作の 4 つのノードの種類がある。対応付けされたノード同士のノードの種類が異なっていた場合、該当ノードにコメントを付加する。

・子ノードの有無が一致しない

対応付けされたノード同士の子ノードの有無を確認する。一方のノードにのみ子ノードが存在する場合は、「ノードが定義されていない」あるいは「余分なノードが定義されている」旨のコメントを付加する。

コメントが付加されているノードについては、そこで再帰的なコメント付加処理を終了する。コメントが付加されていないノードについては、当該ノードが子ノードを持つならば、子ノードを対象として再帰的なコメント付加処理を継続する。

3.4 コメントが付加されたノードへの色づけ処理

模範解答との一致判定機能で付加されたコメントは、該当ノードを選択すると表示される。そこで、どのノードにコメントの付加が行われたかをわかるようにするためにコメントの付加が行われたノードの色を変更している。

ノードの色づけ処理にはツリーコントロールのカスタムドローを使用し、ノードにコメントが付加されている場合はノードの背景色をピンク色にして描写を行うように設定する。一致判定機能でコメント付加が終了した後、ツリーコントロールを再描写することで即時に色付けを反映している。

4. 模範解答編集ツール Pras.Edit

4.1 目的

模範解答との一致判定機能で利用する模範解答には添削用のコメントを保持しておく必要があるが、node-id と review-comment は拡張機能に使用する情報なので保持しておく必要がない。また、今後も機能の拡張により模範木構造のデータ構成が変更される可能性もある。そこで、Perseus における設計木構造と区別するために、模範解答をデータ構成の異なる木構造として作成する。模範木構造は設計木構造と同様に DOM を利用した木構造を用いて表現する。模範解答の各ノードは、正解を保持する right-answer、対応するノードがなかった場合の添削用コメントを保持する error-message の 2 つの情報を持つ。これに伴い、Perseus とは別のツールとして模範木構造を編集するための模範解答編集ツール Pras.Edit の開発を行う。

Pras.Edit は Perseus をベースとして開発しており、操作対象を模範木構造とする。また、ログ機能や PDF 出力機能等、模範解答作成に不要な機能を省略し保守作業を行いやすくしている。また、図 3 に示す UI を持っており Perseus と同様の操作で模範解答の編集を行うことができる。

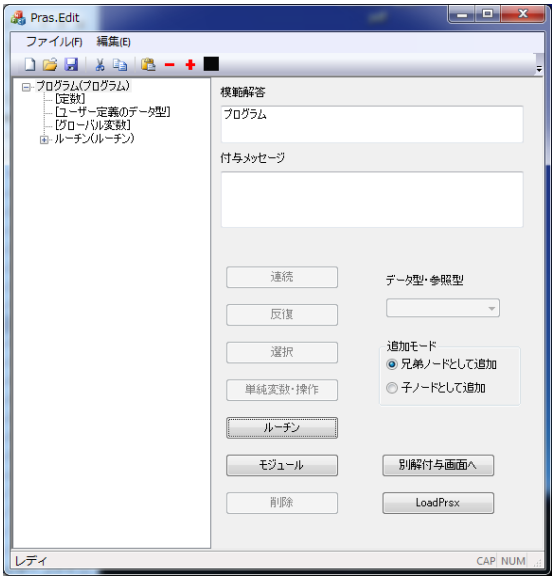


図 3　Pras.Edit のユーザインターフェース

4.2 機能

Pras.Edit は模範木構造の編集機能と模範解答の作成を支援する機能を提供する。

模範木構造構の編集

Pras.Edit の模範木構造編集機能では Perseus で提供している設計木構造の編集機能と同じものを提供する。操作対象ノードを選択し、ノードに編集操作を加えることで模範木構造の設計を行う。また、ノードを選択後に UI 上の下のテキストボックスを編集することで添削用コメントを付加できる。Perseus の設計木構造と区別するため、保存、読み込みを行う模範木構造の拡張子は prasx としている。

設計木構造の変換機能

設計木構造と模範木構造でデータ構成を変更しているため、データ間には互換性がない。そのため、Pras.Edit では設計木構造の読み込みや編集作業はできない。その結果、Perseus で作成された設計木構造を模範木構造に変換する際、手作業で複製しなければならず手間がかかる。そこで、設計木構造を模範木構造に変換する機能を実装した。

設計木構造を読み込むと、まずルートノードのみの模範木構造を作成する。模範木構造のルートノードと設計木構造のルートノードを対応付け、再帰的に子ノードを複製することで同様の設計をもつ木構造を再構築している。子ノードの複製時にはノードの設計情報である comment ノードを、解答を保持する right-answer ノードとしてコピーする。設計木構造の node-id と review-comment は、模範解答に必要ない情報のため破棄する。

模範解答の暗復号化機能

模範解答との一致判定機能は現在、教員が設計木構造を添削する際に使用しているが、将来的には学生がソフトウェア設計を行う過程で使用することを検討している。その場合、学生が模範解答を直接見ることができないように、模範解答を暗号化する必要がある。これに対応するための模範木構造の暗号化、復号化機能を提供する。暗号化の有無の情報は模範木構造に保持しており、暗号化と復号化は同じボタンを押すことで実行できる。また、暗号化しているときは模範木構造の編集ができないようにしている。

現在設計している模範木構造の文字列を保持するノードがあった場合、その文字列を暗号化及び復号化する。ルートノードから末端ノードまでを再帰的に探索し、木構造の全てのノードに対して処理を行うことで木構造全体の暗復号化を行っている。暗復号化には外部ライブラリとして「cryptlib」を用いている。暗号化したデータは XML では直接保持できないため、暗号化した際のデータは Base64 型へ変換した状態で保持している。

4.3 別解機能

模範解答との一致判定機能は文字列のレーベンシュタイン距離を利用して添削を行っている。そのため、表記の揺れが発生しているノードに対して正しい判定が行われないことがある。表記の揺れとは、同じものを意図して設計されたものであっても学生の記述と教員の記述で表現が違うために発生するものである。このとき、二つの文字列間では文字列の類似度が低いためレーベンシュタイン距離が大きくなる。そのため、意図したノード同士のマッチングが行われない、またマッチングしたとしても余分なノードとのマッチングだと判定される可能性が高くなる。

表記の揺れに起因する判定ミスに対応するために、模範解答の各ノードに別解として複数の解答を持たせる機能を導入する。これを用いて表記の揺れに該当する文字列を別解として持たせ、全ての解答の中からレーベンシュタイン距離が最も小さいものを選ぶことで意図したノード同士の対応付けが行われやすくなる。また、別解をマッチングに反映させるために、ノード同士の対応付けの類似度表作成時のレーベンシュタイン距離の計算方法にも修正を加える。模範解答が別解を持つ場合全ての解答とのレーベンシュタイン距離を計算し、最小のものを要素とすることとした。なお、別解は模範解答一致判定機能で出力されるマッチング結果を解析することにより最適なものを選ぶことができると考えられる。

模範木構造のデータ構成の変更

現状の模範木構造のデータ構成では別解を付加することができないため、図 4 のように別解を保持するための alternate-answers を追加する。該当データに子ノードとして alternate-answer ノードを追加することで複数の別解を保持することもできる。

```
- <simple-statement expand="true">
    <right-answer>成績一覧:指定した科目の得点がNULLの学生一覧を返す.</right-answer>
    <error-message>得点未入力の名簿を取得するルーチンが不足している</error-message>
  - <alternate-answers>
      <alternate-answer>成績一覧:指定した科目の得点が入力されていない学生一覧を返す</alternate-answer>
      <alternate-answer>成績一覧:指定した科目の得点が未入力の学生一覧を返す</alternate-answer>
    </alternate-answers>
  </simple-statement>
```

図 4. 模範木構造のノードを表現する XML データ

Pras.Edit への別解付加機能の実装

模範木構造に別解を付加するために Pras.Edit に別解付加機能を実装する。UI は木構造の編集機能とは別に設定し、画面遷移ボタンを押すことにより木構造の編集用 UI と図 5 の別解付加用 UI を遷移する。テキストエディタに別解を入力し、追加ボタンを押すと選択しているノードに別解を付加する。選択しているノードの持つ別解はリストボックスに表示され、別解を選択し削除ボタンを押すことで削除することもできる。また、定数等の固定ノードに対しては別解を付加できないようにしている。

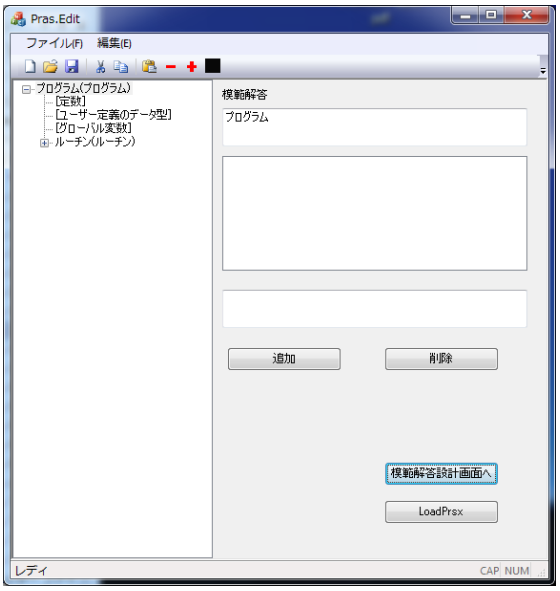


図 5. 別解付加画面

5. 模範解答との一致判定機能の評価実験

模範解答との一致判定機能を用いることにより、添削の自動化を行い作業効率の向上を図ることができる。しかし、本機能ではレーベンシュタイン距離を用いているため、設計ミスの見逃しや正しいものを誤って検出する可能性がある。これらの添削ミスを少なくすることで本機能の有用性を高めることができる。そこで、現時点での本機能の有用性を示すために一致判定機能を使用した際のミスの検出状況を調査する。

5.1 評価実験について

評価実験には知能情報システム学科の学部生 20 名がソフトウェア工学の演習課題として提出したモジュール設計問題の設計木構造を使用する。これらのソフトウェア設計に対して教員の作成した模範解答を用いた一致判定機能を使用し、設計木構造の点検を行う。

本機能を用いた場合の点検結果と、教員が手作業で点検した場合の結果を比較する。比較を行うために、ソフトウェア設計に対して模範解答との一致判定機能で検出したミスを以下の 4 つの場合に分類する。

A) ツール、手作業のどちらでも検出したミス
B) ツールでは検出できず、手作業では検出したミス
C) 手作業では検出できず、ツールでは検出したミス
D) ツールで誤って検出したもの

模範解答との一致判定機能ではノード一つ一つにコメントの付加を行うが、手作業での点検ではノード単体ではなく設計全体に対する指摘が示されている。また、模範解答との一致判定機能でのコメントは不足しているノード、及び余分なノードを指摘する簡潔なものとなっている。そのため、模範解答一致判定機能でのコメントが手作業での添削内容と一致していると判断されるものは A と分類し、そうでないものは C と分類している。手作業での添削コメントの総数は 72 個となっており、ツールでの添削コメントにおいて一致するコメントが一つも無かった場合を B としている。また、模範解答一致判定機能における添削では正しいノードに対して誤って添削することがあるため、これを D に分類している。

また、新機能として実装した閾値の変更機能と別解機能を用いることで、どの程度、判定精度を上げることができるかも調査する。以下でこれらの実験結果と考察を述べる。

5.2.1 一致判定の結果と考察

まず、レーベンシュタイン距離を用いた一致判定の閾値は初期値の 15 にし、別解は使用しない状態での判定結果について確認する。表 1 がこの場合における判定結果の一覧表である。一致判定機能で検出した 1189 個のミスのうち誤りは場合 D の 259 個であることから、誤検出率は 21.8%である。

表 1 実験結果

学生	A	B	C	D	総計
1	29	2	23	5	59
2	16	0	35	22	73
3	32	0	9	4	45
4	4	0	24	14	42
5	11	0	35	11	57
6	32	0	27	16	75
7	11	0	40	11	62
8	61	0	40	22	123
9	20	0	37	8	65
10	11	0	19	11	41
11	13	0	26	13	52
12	30	0	21	20	71
13	15	1	14	18	48
14	26	1	21	20	68
15	13	0	40	13	66
16	11	0	27	8	46
17	6	0	42	5	53
18	10	0	38	10	58
19	0	0	25	8	33
20	7	0	29	20	56
総計	358	4	572	259	1193

手作業による添削コメント 72 個のうち、該当するコメントが自動検出されなかったものは場合 B の総計に示したとおり 4 個であり 5.6%程度に留まる。見逃した 4 つのミスとしては、ルーチンノードの段階でマッチングが不成立となったため、その下位にあるデータ構造部の点検が行われなかったものと、余分なノードとのマッチングが成立したため点検が行われなかったものの 2 通りがあった。

一方、一致判定機能を用いることで手作業では発見でき

なかった誤りを合計 572 個検出できたのは、本機能の有用性を示す結果である。

また、全体の添削コメントに対して誤った添削を行っているもの（場合 D）の割合は 22%である。本機能のマッチングの成立判定はノード同士のレーベンシュタイン距離と設定した閾値の比較により行っている。そのため、誤った判定を行う理由としては以下の 3 点が挙げられる。

1. 正しいノード同士のマッチングにおけるレーベンシュタイン距離が閾値を上回ることによる誤検出
2. 余分なノード同士のマッチングにおけるレーベンシュタイン距離が閾値を下回ることによる見逃し
3. 意図したノードとのマッチングが行われていないことによるマッチングミス

評価結果に基づいて誤判定が行われる原因を調査してみたところ、大きく分けて 4 つの原因があった。

1 つ目は学生の記述と教員の記述が同じ意図をしたものでも違う記述になっている表記の揺れによるものであり、上記の 1 と 3 の主な発生原因である。これにより対応したノードのレーベンシュタイン距離が大きくなり、マッチングが未成立と判断される、あるいは意図したノードとのマッチングが行われなくなるために判定ミスが発生している。今回の判定ミスにおける最大の要因となっており、場合 D に該当する判定ミスの約 7 割がこれに起因する誤検出であった。

2 つ目は誤って定義されたノードが対応付けされたときに、文字列の類似度が高いためマッチングが成立してしまう場合であり上記 2 の主な発生原因である。判定ミスの 2 割がこれに起因する誤検出である。「指定した学生の～を返す」といったルーチンが誤って定義されていた場合に、一部分のみしか文字列が違わないために発生することがある。また、文字列が短い場合にレーベンシュタイン距離が小さくなることにより発生しているものも少数ではあるが発生している。

3 つ目は戻り値や引数の定義が間違っているために正しいノードとのマッチングが成立していない場合である。ルーチン設計部におけるノードの対応付けでは、ルーチン名に加えて戻り値と引数の部分までを類似度表に反映するようにしている。そのため、戻り値や引数が大きく間違っている場合に他のノードとの対応付けがされてしまっている事例が確認された。なお、これを要因とする判定ミスは全体の 2%程である。

4 つ目はルーチンの処理の仕方が教員の作成したものと異なる方法で設計されている場合である。これに起因する判定ミスは約 10 個で全体の 4%程である。この場合、表記が一致しないため対応付けや判別は困難であり、マッチングが成立したとしても戻り値や引数等が違うものになるため適切な添削ができない。これについては別途対応策を検討する必要がある。

5.2.2 閾値を大きくした場合のマッチング結果と考察

前節の条件でマッチングを行った場合、レーベンシュタイン距離が閾値を上回ることによりマッチングミスにより添削ミスが多くなっていた。そこで、マッチングの成立判定に用いる閾値を大きくすることで判定精度が向上する可能性がある。また、ルーチンノードにおけるマッチングの出力結果を見たところ、誤検出が起きているノードのレーベンシュタイン距離は 20 を下回っているものが多かった。そこで、閾値を 5 大きくした 20 として再度同一のデータに対して検証を行った。閾値を変えた際の判定結果では、場合 A が 345 個、場合 B が 3 個、場合 C が 602 個、場合 D が 225 個検出された。この場合の誤検出率は 19.2%である。

閾値を 15 として判定した結果と比較すると、誤った判定を行っているものは 1 割ほど減少しているが、正しく判定できている場合 A と C の合計個数はあまり変化していない。さらに、場合 A の判定となっているノードの数が減少していることがわかる。これは、閾値が大きくなったことにより正しく添削できていたノードを見逃したのが原因である。

誤った添削を行ったノードの内訳を確認してみたところ、閾値を 15 とした時と比較して正しいノードとのマッチングが不成立になっているノードの数は 5 割ほど減少している。予想より減少量が低かった理由として、ルーチンノード部の表記の揺れについては概ね対応できるようになっていたが、アルゴリズム部のルーチン呼び出しにおいては更に表記の揺れが大きくなっているものがあり不成立となっているものが新たに発生していた。また、余分なノードとのマッチングが成立したために誤った判定が行われていたものが 5 割ほど増加していた。余分なノードとのマッチングにおいて、一部分の単語だけがあっているものや設計側の文字列が短いもの等がマッチング成立状態になってしまっていたためであった。場合 A の判定数が減っていたのはこれが原因であり、正しくミスを指摘できていたノードを見逃すようになっていた。結果として、閾値を変更する前と比べても結果にはあまり差がないことがわかった。

5.2.3 別解を付加した場合のマッチング結果と考察

閾値を変える前と変えた後の二つの添削結果やマッチング結果を調査したところ、ルーチンの記述に対して同じような表記の揺れにより正しいノード同士のマッチングが不成立になっているものが複数存在することが確認できた。そこで、別解機能を用いて模範解答のルーチンの記述及びアルゴリズム部のルーチン呼び出しの記述に対応している一部のノードに対して別解を付加する。別解を付加した模範解答を使用した際の添削結果では場合 A が 379 個、場合 B が 4 個、場合 C が 664 個、場合 D が 174 個検出された。なお、閾値は 1 回目と同じ 15 としている。誤検出率は 14.3%

に改善された。

別解を付加していないときと比較すると、誤った判定を行っているものは 3 割近く減少しており、正しい判定が行えているものが 1 割ほど増えている。正しい判定が増えている理由としては、ルーチン部で誤った判定を行っているものが減ったためにアルゴリズム部やデータ構造部の判定が行われるようになったことがあげられる。

正しいノードとのマッチングが不成立になっているノードの数は別解を付加する前と比べて半分になっていた。別解を付加したノードだけを見てみると誤って不成立とされていたノードは 2 割まで減っていた。また、表記の揺れにより意図したノードとのマッチングになっていなかったノードについても正しいノードとのマッチングに修正されているようになっている確認できた。別解を付加したことにより余分なノードとのレーベンシュタイン距離が小さくなり誤った判定を行っているものも発生していたが、これは 4 つほどの極めて少ない数であった。

今回付加した別解は学生の記述における表記の揺れを参考にしている。ただし、「成績の記入」といった文字列が短いものについてはそのまま別解として保持してしまうと誤検出の要因となる可能性がある。そのため、該当する表記については文字列が短くならないように「指定した学生の成績を記入する」等の記述に改めたものを別解として付加した。この結果から、表記の揺れが多く見られる記述に対して適切な別解を付加することで該当ノードのマッチングミスを大幅に減らすことができることがわかった。

6. 関連研究

ソフトウェア設計を支援するツールは既存のものとして PAD や UML が存在する。PAD ではアルゴリズムを記述できるがデータ構造の設計は行えず、UML はモジュール間の関係やデータ構造を詳しく記述することができるが、アルゴリズムを書くことができない。Perseus ではこれら全ての設計に対応している。

また、設計ミスを自動で検出する手法として UML 設計図からメトリクス値を算出し、ソフトウェア設計の欠陥を検出する方法が提案されている[7]。手法は UML 設計をいくつかの工程にわけ、各段階でメトリクスを計算しその値から欠陥検出を行うものである。

本研究と異なる点は、対応している設計が異なる点が挙げられる。UML ではアルゴリズムの記述が行えないが、Perseus では可能なためそれについてもミスの検出をすることができる。また、ミスの検出方法においても異なっており各設計のルールに基づいて設計ミスの検出を行っており、記述のチェックは行っていない。本研究では模範解答との比較を行い記述のチェックを行っている。これにより、設計ミスをより詳細に指摘できる。

7. おわりに

本論文では Perseus における模範解答との一致判定機能の開発について述べ、評価実験を行った。本機能を用いることで答案の点検を自動化することで教員の作業効率を高めると同時に、より詳細な誤りの指摘も行えるようになった。

模範解答との一致判定機能ではレーベンシュタイン距離を用いてノード同士の対応付け及び対応したノード同士の成立判定を行っている。そのため、教員の記述と学生の記述に表記の揺れがあると正しい添削が行われないことがあった。そこで、表記の揺れによる添削ミスを低減するために閾値の変更機能と模範解答の別解機能を実装した。別解機能の実装に伴い模範解答を編集するためのツールである Pras.Edit の開発を行った。また、模範解答一致判定機能の有用性を示すために評価実験を行い、実装した機能を利用した場合と利用しなかった場合を比較し考察を行った。結果として、模範解答との一致判定機能ではミスの見落としは少ないが誤って添削を行っているものが全体の 2 割ほどに上っていた。追加した機能を利用した場合、閾値を変更してもあまり効果は上げられなかったが適切な別解を付加することにより誤って添削しているものを減らすことができた。

今後は評価実験で得られた添削結果を元に、どのように選ぶことでより適切な模範解答の別解を作ることができるかの分析を行う。また、別解機能では余分なノード同士における添削ミスやルーチンの動作の違いにより発生する添削ミスには対応できないため、これらに対する対策を考える必要がある。これらの作業が完了したら再度評価実験を行い、改めて模範解答一致判定機能の添削内容が改善されているか調査する。

参考文献

1) 独立行政法人 情報処理技術者推進機構、技術本部ソフトウェア・エンジニアリング･センター：共通フレーム 2013，独立行政法人 情報処理技術者推進機構，2013.
2) 井上克郎：演習で身につくソフトウェア設計入門，株式会社エヌ・ティー・エス，2006.
3) T. Kakeshita, T. Fujisaki, “Perseus: An educational support tool for systematic software design and algorithm construction”, Proc. CSEE&T, 2006.
4) 柴田祐貴，山下智史，掛下哲郎：ソフトウェア設計支援ツール Perseus における模範解答との比較機能, 電気関係学会九州支部連合大会，2014.
5) 山下智史, 柴田祐貴, 掛下哲郎：要求管理教育支援ツール REMEST における模範解答を用いた検査機能の開発, 電気関係学会九州支部連合大会，2014.
6) 久保 龍彦：diff の動作原理を知る～どのようにして差分を導き出すのか，Software Design，2009. http://gihyo.jp/dev/column/01/prog/2011/diff_sd200906
7) 影山智一，上田賀一：初学者を対象とした段階的 UML 設計手法の提案，情報処理学会研究報告：ソフトウェア工学（SE），1-8，2014.