

Automated Extraction of Domain Models From Textbook Indexes For Developing Intelligent Tutoring Systems

Rabin Banjade¹, Priti Oli¹, and Vasile Rus¹

University of Memphis, USA
{rbanjade1,poli,vrus}@memphis.edu

Abstract. Domain modeling is an important task in designing, developing, and deploying intelligent tutoring systems and other adaptive instructional systems. We focus here on the more specific task of automatically extracting a domain model from textbooks. In particular, this paper explores using multiple textbook indexes to extract a domain model for computer programming. Our approach is based on the observation that different experts, i.e., authors of intro-to-programming textbooks in our case, break down a domain in slightly different ways, and identifying the commonalities and differences can be very revealing. To this end, we present automated approaches to extracting domain models from multiple textbooks and compare the resulting common domain model with a domain model created by experts. Specifically, we use approximate string-matching approaches to increase coverage of the resulting domain model and majority voting across different textbooks to discover common domain terms related to computer programming. Our results indicate that using approximate string matching gives more accurate domain models for computer programming with increased precision and recall. By automating our approach, we can significantly reduce the time and effort required to construct high-quality domain models, making it easy to develop and deploy tutoring systems. Furthermore, we obtain a common domain model that can serve as a benchmark or skeleton that can be used broadly and adapted to specific needs by others.

Keywords: domain modeling · computer programming · intelligent tutors system · computer science education · Java programming

1 Introduction

When developing an intelligent tutoring system (ITS), it is imperative to understand what students need to master in the target domain. This requires an effective representation of the domain, which includes the key concepts or knowledge components (KCs) and a structure that specifies a prerequisite structure over those KCs. This representation of the domain is the domain model, and it plays a key role in ITSs. Indeed, these intelligent tutors rely on the domain model to select what tasks the student should work on and to provide relevant support,

e.g., scaffolding in the form of hints or correcting a misconception when students work on a task. While our main goal was to develop an ITS for code comprehension, we learned that there is no off-the-shelf, ready-to-use domain model for intro-to-programming topics typically covered in CS1 and CS2 courses. To the best of our knowledge, there is no domain model for intro-to-programming that covers the vast majority of topics and is at the right level of granularity for our purposes. Furthermore, there is no widely accepted domain model that anyone can reference and use as a baseline [21, 13].

In sum, the work in this paper was driven by two main reasons: (i) our broader goal of developing an Intelligent Tutoring System (ITS) for intro-to-programming and (ii) the need for a domain model for intro-to-programming that is publicly available and ready to use, i.e., it can serve as a benchmark or skeleton that can be used broadly and adapted to specific needs by others.

Typically, the design and evaluation of the domain model for a given domain is expert driven, which is both tedious and expensive. Such manual approaches hinder the development of ITSs for new domains and, therefore cross-domain scalability of ITSs. Automating the extraction of domain models will address these challenges. To extract domain models, various sources can be used, such as textbooks that represent authors' perspectives of the targeted domain [17]. It should be noted that the authors are both experts in the domain and pedagogical experts, i.e., they do have experience teaching about the target domain, which makes them familiar with, for instance, the typical misconceptions and more effective ways to teach about a particular concept. Thus, extracting domain models from textbooks is based on solid expertise.

Textbooks are mostly free text but contain other, more structured elements such as table of contents, chapters, sections, etc. Of particular interest to our work is the index at the back of a typical textbook which serves as an important navigational tool that links the important domain terms curated by the expert to specific parts of the textbook. That is, entries in such textbook indexes are a rich source of domain terminology. The index at the end of a textbook is created mostly manually, even though some automated methods have been studied [24]. Index terms are not just a collection of words but a reference model based on predefined rules. Although index terms can be a source of curated domain terms, from an epistemological perspective, an index of any document reflects not only the expertise and efforts of its creator but also the needs of the group of users for whom the index is created and the task that these users engage in [9]. This can lead to index terms that might not be fully representative of the domain. Hence, extracting the domain model from the textbooks using the indexes may not necessarily lead to a clean domain model. Also, not all the index terms are useful for learning or teaching and may contain various useful entries about the history of a domain. Such examples, in our case, ranged from method names to names of persons (e.g., Gosling James, the creator of Java). If the goal is to create a reference domain model that includes concepts that everyone agrees on, a good approach would be to use the intersection of concepts from multiple

books i.e. including concepts that all the authors specify and excluding concepts unique to a single textbook.

Indeed, our approach is to use the intersection of index terms from multiple textbooks to extract a common domain model for intro-to-programming. This intersection will create a common skeleton which can then be augmented with author/expert-specific concepts. A union approach, i.e., the union of all the concepts specified in the index terms lists of various textbooks, will lead to a more comprehensive domain model that would include many author-specific concepts. We plan to explore such an approach in the future. Here we focus on discovering the common domain model across many textbooks.

It should be noted that performing the intersection or reunion of index terms lists in an automated way is not a straightforward task, given the various ways authors may specify the same underlying concept. For example, the concept of ‘two-dimensional arrays’ could be represented in a hierarchical way, with level 1 specifying ‘array’ and level 2 ‘two-dimensional’, which can be conflated together to get ‘arrays- two-dimensional’. To address this issue, we investigate the use of various approaches based on concept similarity methods (perfect string matching, approximate string with word-to-word semantic similarity) and majority voting, i.e., a concept should be present in the majority of the textbooks used, e.g., in two textbooks out of three textbooks considered. To identify common concepts across three books, we use TFIDF (term frequency-inverse document frequency at character level as detailed later), and transformer-based pre-trained language models for approximate string matching [7].

The major contributions of this paper are 1) automated approaches to extract a common domain model using index terms at the end of textbooks and 2) evaluation of these approaches with respect to a gold standard domain model generated by experts from the same textbook indexes.

The outline of the paper is as follows. The next section, *Related Work*, briefly highlights key prior efforts in the automated extraction of domain models and automatic domain model refinement. It also covers previous efforts on general techniques for approximate string matching. The *Approach* section outlines the key steps of the proposed approach to domain modeling extraction from intro-to-programming textbooks. Then, we present details about our experiments and the results obtained. The *discussion and Future Work* section highlights the important aspects of the proposed approach, contributions, and plans for future work.

2 Related Work

2.1 Domain modeling

Previous work in domain modeling extraction has explored various approaches and extracted domain models from various sources. Many such previous works rely on three main information sources: experts in the domain, learners’ data, and textbooks. Many previously proposed approaches are based on keyphrase

extraction, and content analysis [2, 4, 1, 3]. One previous effort similar to our work was proposed by Zechmeister and colleagues [25] who manually analyzed ten psychology textbooks to create a set of core concepts. Our work is a step towards automating this approach. In the context of the domain model for computer programming, although works such as [21] provide guidelines for conceptual content to be covered in introductory computing courses, it does not provide a fine-grained list of concepts needed for ITSs. Efforts to extract fine-grained concepts using multiple textbooks were reported [22]; however, these efforts were manual and difficult to scale. Some automatic efforts such as [11, 20] relied on Java parsers or Abstract Syntax Tree (AST) to extract concepts related to a particular computer program. Our work which is done in the context of CS1 and CS2 computer programming courses can provide a fine-grained domain model based on textbooks and can be used in other domains as well.

2.2 String Matching algorithms and pretrained transformers

Using exact string matching for our purposes is too constraining, as illustrated later because authors tend to refer to the same concept using different linguistic forms. Therefore, to identify commonalities, we must use more flexible matching methods.

Approximate string matching has been explored in various applications including address matching, name matching, biomedical abbreviation matching as well as spelling correction [5, 19]. The use of such string-matching techniques and algorithms has not been explored previously for domain modeling to the best of our knowledge. Approximate string matching [16] has been explored in many different variants; for example, Cohen et al. [5] experimented with edit-distance, token-based distance and hybrid methods, Ji and colleagues used BERT for biomedical entity normalization [12]. Despite extensive work on fuzzy or approximate string matching, there is no consensus on what method works best and under what circumstances. For our task of matching similar index terms across different textbooks, we consider two main approaches based on TF-IDF and pretrained transformers. The former approach takes into account surface form similarity of index terms based on a character-level analysis whereas the latter takes into account semantic similarity based on pretrained neural embedding models. We experimented with BERT and its variation PhraseBERT [23] which is specifically fine-tuned to capture lexically diverse phrases.

3 Dataset

3.1 Sample of textbooks and Index terms

We used three textbooks to develop and evaluate the proposed automated methods for domain modeling: Introduction to Java [6], Bigjava [10], and Java-How to Program [18]. These textbooks are used as introductory Java programming textbooks but also contain advanced concepts such as networking, java database programming, etc. We focused on index terms from the chapters commonly taught

in CS1 and CS2 courses [22]. The indexes of these textbooks generally include entries or index terms, followed by locators, which can be page numbers, page number ranges, or section numbers. Some indexes contain semantic relationships with other index terms, such as subcategories, synonyms, or cross-references. While there are guidelines for creating an index, there can be organizational and content variations among publishers, domains, books, and authors.

From the perspective of our work here, we took advantage of the hierarchical structure for the indexes of the three textbooks, i.e., indexes in those textbooks describe entries using two hierarchical levels: first/top level and second level. Examples of this hierarchical structure in the three textbooks are shown in Figure 1.

Floating-point numbers assigning to integer variables, 134 comparing, 188 converting to integer, 142–143. <i>See also</i> cast operator. description, A-72 mixing with integer, 139 precision, 133–134	abs method, Math class, 121–122, 524 Absolute file name, 473 Abstract classes AbstractCollection class, 762 AbstractGraph , 1025–1026 AbstractGraph.java example, 1028–1033 AbstractMap class, 812 AbstractSet class, 798 AbstractTree class, 935–936	argument list 1398 argument promotion 207 argument to a method 41, 76 arithmetic and logic unit (ALU) 9 arithmetic calculation 53
---	--	---

Fig. 1. A snippet of index terms in three different books

The second-level index terms are related to the first/top level, but if the second-level index terms were important, they were also present as first-level entries. For example, for the first-level index term “Array,” the second-level index terms might be “sorting,” “for each loop,” “declaring,” and “accessing elements.” Out of these second-level index terms, “for each loop” and “sorting” were also present as separate first-level entries, but “declaring” and “accessing elements” were not. In this paper, we focused on the first-level entries as it seems to be a good start to create a common domain model. We obtained 1495, 2090, and 875 first-level index terms from three books.

3.2 Data Annotation

We created a gold standard domain model for computer programming by annotating a sample of index terms that we considered for our analysis. To achieve this goal, we recruited three graduate students well-versed in computer programming to perform the annotation process. The index terms were annotated based on their importance from a pedagogical perspective as judged by the graduate student. They thus represented a computer programming concept worth learning for novices. Our annotators annotated the union of index terms from three Java programming books. As stated earlier, our annotators annotated 4460 first-level index terms. To ensure that our annotations were not specific to these books, we only marked those terms as concepts that were not specific to the book, e.g., we excluded terms that represented method names, classes, or Java-specific terms and only marked those terms as concepts that were generalizable to other programming languages.

We evaluated the inter-rater agreement of our annotation process and found that it was considerable with a pairwise agreement of 0.82, 0.83, and 0.72. However, to ensure the accuracy of our domain model, we only considered index terms as domain terms to which all annotators agreed after the first round of annotation. This scheme enabled us to create a consensus domain model for computer programming. We obtained 465 terms out of 4,204 concepts in the union of index terms from all three textbooks.

The next step in creating the gold standard domain model was concept normalization or canonization, which means identifying different phrasings of the same concept and marking them as referring to a single concept for which a canonical form should be adopted. We call such groups of terms representing a single concept a synset (synonymous set) similar to the set of synonymous words in WordNet [15] that describe the same concept. For example, “binary digits” and “bits” represent a synset. We grouped terms together only if they represented exactly the same concept at the same level of generality or specificity. For example, “for loop” and “while loop” represented two different synsets. As a result, we obtained 263 concepts out of 465 terms. This gold standard domain model will be used to guide our further analysis of computer programming concepts. We use this gold standard domain model to evaluate our results.

4 Approach

We experimented with different approaches for extracting domain models based on index terms and using automated approaches. The first approach, *Consensus voting with perfect string matching*, identifies index terms that are common across all the three books in exactly the same linguistic form. For example, concepts such as ‘recursion’, ‘inheritance’, and ‘infinite recursion’ were present in all three books. The second approach, *majority voting with perfect string matching*, relaxes a bit the prior approach and only requires exact terms be present in at least two textbooks, one such example is ‘infinite loops’ which is mentioned in two indexes of two of the textbooks but not all three. Furthermore, our third approach, *majority voting with approximate string matching*, tries to identify similar concepts such as (‘abstract data type (adt)’, ‘adt (abstract data type)’) expressed linguistically in different ways in different books. Another example of a concept expressed in different ways is ‘bits’ and ‘binary digits’. Even though these two phrases refer to the same concept, a string-based method cannot identify them as similar. However, models such as BERT can capture the semantic similarity between these terms. We explain each method in this section.

4.1 Consensus Voting with Perfect String Matching

Let A, B, and C be the lists of index terms representing index terms from each of the three textbooks and t a specific string describing a concept. Let D be the list of identical terms in all three textbooks. Considering D as the domain

model, we can represent domain model D obtained using consensus voting with perfect string matching as:

$$D = t \mid (t \in A) \wedge (t \in B) \wedge (t \in C) \quad (1)$$

4.2 Majority Voting with Perfect String Matching

Majority voting with perfect string matching represents a set of terms present in at least two sets of index terms. As in Equation 1, let D be the list of terms present in at least two of the three indexes: A , B , and C . We can then represent domain model D as:

$$D = t \mid (t \in A \wedge t \in B) \vee (t \in A \wedge t \in C) \vee (t \in B \wedge t \in C) \quad (2)$$

The equation ensures that a term must be present in the same linguistic form (identical string) in at least two out of the three indexes to be included in D . This criterion represents majority voting, as it ensures that a term is only included in D if it has majority support from the textbook indexes.

4.3 Majority Voting with Approximate String Matching

Considering the list of index terms from each book A , B , and C and D as the resulting domain model, Let t and t' be two terms in A , B , and C such that they belong to different indexes and $sim(t, t')$ be the similarity between the terms t and t' based on matching approaches. Let τ be a threshold value for similarity. The domain model is thus the following:

$$D = t \mid sim(t, t') > \tau \wedge ((t \in A \wedge t' \in B) \vee (t \in A \wedge t' \in C) \vee (t \in B \wedge t' \in C)) \quad (3)$$

This equation states that a term t is in the domain model D if its similarity with any other term t' in another list is higher than the threshold τ . This represents the majority voting, as a term must be similar to terms in the majority of lists to be included in the domain model D . We experimented with two term similarity methods: TF-IDF at the character level, which is more of an approximate string matching method capturing morphological variations of two index entries, and pre-trained transformer-based models from which we extract embeddings of index terms that can capture the semantic similarity between two index entries. Approximate string matching using TF-IDF and pre-trained transformer models were implemented using freely available python library[8].

Approximate String Matching with TF-IDF: We opted for TF-IDF-based string matching to calculate the string similarity between two index entries based on the cosine similarity of TF-IDF of character level n-grams (n=3) of index terms. We select the index term from another book with the highest similarity score for each index term in a book. A similarity score of 1 indicates that the terms match perfectly, whereas 0 indicates the index terms do not have any n-grams in common. Even though 0 and 1 thresholds on similarity give minimum

and maximum bounds, we need a data-driven threshold value that maximizes the accuracy of detecting index terms that represent the same concept. Our threshold selection approach is explained in further detail in section 4.3. After thresholding, we obtain pairs of index terms representing similar concepts that may have different forms in different books. TF-IDF-based approximate matching captures index terms that vary on character level n-grams, such as: ‘string concatenation’ and ‘concatenate strings.’

String Matching with Pretrained Transformer models: Even though token-based string matching can capture the differences in the token level of the index terms, such as inflections or different forms, i.e., orthographic variations, they are limited in expressiveness and fail to capture similar concepts based on their semantic similarity, i.e., index terms that use different wordings to represent the same concept. To overcome this limitation and accurately measure the semantic similarity of index terms, we conducted experiments with two transformer-based models: BERT and PhraseBERT.

For instance, the index terms “bound checking” and “bound errors in arrays” refer to the same concept of bound errors in arrays, but a traditional n-gram method like TF-IDF might treat them as different due to the lack of shared words. In contrast, transformer-based models like BERT can consider contextual information and identify that these terms are semantically similar. However, research has shown that BERT relies more on lexical overlap to determine semantic similarity [26, 14], which can result in less diverse phrases in phrase-level semantic-relatedness tasks. To address this issue, we also experimented with PhraseBERT, which fine-tunes BERT using contrastive learning to produce more robust and diverse phrase embeddings.

We extract the embedding of each index term by averaging the embeddings of tokens obtained from pretrained transformer models representing the index terms. We calculate the cosine similarity of obtained index term embeddings to find the best match.

Selecting approximate threshold: Our string matching techniques estimate the similarity between two index entries based on the cosine similarity values and a threshold value above which the two index entries are considered similar, i.e., referring to the same underlying concept. We need the optimal threshold value: a higher threshold will result in fewer matches being identified but likely to be more precise at the expense of lower recall, whereas a lower threshold will result in more matches but with less precision. Since we are using cosine similarity, a score of 1 would give the exact matching index terms but would ignore similar terms using different words. To select the appropriate threshold for each of our methods for approximate string matching, we searched over the set of possible threshold values (between 0 and 1) using an increment step of 0.1. The goal was to identify the threshold value that maximizes the F-score, which is the balance of precision and recall. To calculate precision and recall and the F-measure, we manually annotated a sample of concepts as annotating all

concepts was too prohibitive. We used stratified sampling with proportionality to ensure that our validation set consisted of a representative sample of each threshold value based on its proportion to the overall sample size, which in our case represents similarity scores of each pair of indexes. This allowed us to obtain accurate precision and recall values without annotating all the samples. Based on this search maximizing the F-measure, we selected 0.8 as the threshold for TF-IDF-based methods and 0.9 for BERT-based methods. This allowed us to choose appropriate thresholds that balanced precision and recall and maximized the F-score. Overall, this approach enabled us to select appropriate thresholds for string-matching techniques efficiently and effectively by maximizing the F-score for the selected thresholds. The plot for precision and recall for various threshold values for TF-IDF is shown in Fig 2.

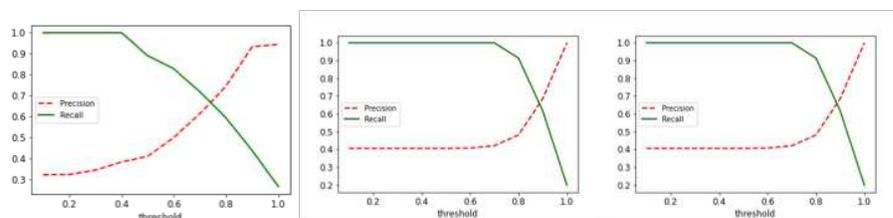


Fig. 2. Precision-Recall against various thresholds for a) TF-IDF b) BERT c) Phrase-BERT based string matching

4.4 Grouping similar terms together

From the approach explained in the previous sections, we obtained pairs of index terms that are similar based on the optimal thresholds. Consider a pair of index terms ('less than operator', '<operator') and (less than operator, logical operator (<)) where the first and second terms in each pair are from different textbooks. As these pairs represent the same concept, which we refer to in canonical form as 'less than operator,' we combine them into a single group representing a concept, i.e., a synset as in our gold standard domain model.

It is to be noted that this grouping depends on the threshold value and string matching algorithm used; in our case, grouping based on TF-IDF consists of terms that are similar in surface forms, whereas pretrained transformer-based models consist of semantically similar terms. For example, *boolean data type* and *boolean expression* were grouped together by pretrained transformer-based models but not by TF-IDF-based model.

5 Results and Evaluation

For each method, we first evaluate the number of terms identified and then compare that to the gold standard. As seen in table 1 extracting common index terms using the exact string matching methods with consensus leads to only 16 terms.

Even though some preprocessing steps, such as stemming, might considerably increase this number by removing morphological variations among words, this small number of exact common terms highlights the challenge of finding commonalities using index terms lists and the fact that different authors/textbooks use different ways to identify and describe the concepts of a domain. When we consider exact matching index terms that are present in at least two glossaries out of three (majority voting approach with exact string matching), we end up with 163 common terms. One might argue that this could be a very good common skeleton of a domain model that many may agree upon, which is indeed a good argument. We intend to release this common skeleton as a good starting point for an intro-to-programming domain model. Using the approximate matching methods (TFIDF, BERT, and Phrase-BERT) to capture common concepts in combination with majority voting across three glossaries, we ended up with 852 terms (using TFIDF), 362 terms (BERT), and 739 (Phrase-BERT). We evaluated those domain models with respect to our gold standard and reported precision, recall, and F-measure.

Table 1. Number of terms obtained for domain model representation using different approaches

Approach	No. of terms
Consensus voting with Perfect String Matching	16
Majority voting with Perfect String Matching	163
Approximate String Matching with TF-IDF	852
Approximate String Matching with BERT	362
Approximate String Matching with PhraseBERT	739

Our gold standard domain model consists of terms grouped into synsets, i.e., terms representing the same concept represent a group. A term in the extracted domain models has a match if it has an exact string match with any term in any synset in the gold standard.

Table 2. Precision, Recall and F1 score for different approaches for domain model extraction

Approach	Precision	Recall	F-Score
Consensus voting with Perfect String Matching	0.5	0.029	0.054
Majority voting for Perfect String Matching	0.34	0.205	0.25
Approximate String Matching with TFIDF	0.26	0.71	0.38
Approximate String Matching with BERT	0.22	0.36	0.27
Approximate String Matching with PhraseBERT	0.22	0.63	0.33

A summary of the results is shown in Table 2. As seen in the table, the highest precision is obtained with consensus voting and perfect string matching, which is understandable since common identical terms across three textbooks are likely to be domain terms. However, not all the terms that were common represented important domain terms. For example, the method identifies ‘*Gosling James*’

as a common term across all three books, which is not a core programming concept. Most of the terms identified by the method refer to syntactic aspects of Java programming, such as keywords in java. Our second approach, majority voting with perfect string matching, leads to higher recall as expected. The TFIDF-based approach had the highest precision and recall of domain terms indicating that most of the domain terms only varied morphologically rather than semantically as indicated by results in BERT and Phrase-BERT. We also noticed that BERT-based methods grouped terms like ‘bitwise shift operator’ and ‘bitwise operators’ together although they are different in the sense that one is a more specific concept than the other which explains the lower recall of those methods.

6 Discussion and Future Work

This paper presents the implementation details and evaluation of different approaches for automated extraction of domain model from the glossary of textbooks. In our case, approximate string matching with TF-IDF gave better results compared to other approaches. This suggests that string-matching techniques that compare lexical differences can capture domain terms better for computer programming, one of the reasons being, the use of specialized jargon and technical terms in programming. Methods that rely on semantic similarity might perform better in domains where a single concept can be represented in many different ways. Nevertheless, considering string matching techniques greatly improves the F1 score. Overall our approaches are promising. One of the concerns during our experiments was grouping the terms while creating a gold-standard domain model. While we resorted to grouping in the most strict sense, grouping only if two terms represent the same concept, this assumption can be relaxed based on the downstream task. Another concern stems from what granularity to consider for domain representation. For example, ‘pass by value’ is an important concept, however, this could also be considered as an underlying concept of ‘function’. As a remark, while our work is mainly based on the first level of index terms, which we sample for core domain model creation, we can extend it further to different levels of index terms to create a hierarchical model showing the relationship among index terms.

ACKNOWLEDGMENTS

This work has been supported by the following grants awarded to Dr. Vasile Rus: the Learner Data Institute (NSF award 1934745); CSEdPad: Investigating and Scaffolding Students’ Mental Models during Computer Programming Tasks to Improve Learning, Engagement, and Retention (NSF award 1822816), and Department of Education, Institute for Education Sciences (IES award R305A220385). The opinions, findings, and results are solely the authors’ and do not reflect those of NSF or IES.

References

1. Alpizar-Chacon, I., Sosnovsky, S.: Knowledge models from pdf textbooks. *New Review of Hypermedia and Multimedia* **27**(1-2), 128–176 (2021)
2. Banjade, R.: Domain model discovery from textbooks for computer programming intelligent tutors. In: *FLAIRS Conference Proceedings*, 34. (2021)
3. Banjade, R., Oli, P., Tamang, L.J., Rus, V.: Preliminary experiments with transformer based approaches to automatically inferring domain models from textbooks. In: *Proceedings of the 15th International Conference on Educational Data Mining*. p. 667 (2022)
4. Chau, H., Labutov, I., Thaker, K., He, D., Brusilovsky, P.: Automatic concept extraction for domain and student modeling in adaptive textbooks. *International Journal of Artificial Intelligence in Education* **31**(4), 820–846 (2021)
5. Cohen, W.W., Ravikumar, P., Fienberg, S.E., et al.: A comparison of string distance metrics for name-matching tasks. In: *IIWeb*. vol. 3, pp. 73–78 (2003)
6. Daniel Liang, Y.: *Introduction to java programming* (2007)
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018)
8. Grootendorst, M.: Polyfuzz: Fuzzy string matching, grouping, and evaluation. (2020). <https://doi.org/10.5281/zenodo.4461050>
9. Hjørland, B.: What is knowledge organization (ko)? *Knowledge Organization* **35** (07 2008). <https://doi.org/10.5771/0943-7444-2008-2-3-86>
10. Horstmann, C.S.: *Big Java: early objects*. John Wiley & Sons (2016)
11. Hosseini, R., Brusilovsky, P.: Javaparser: A fine-grain concept indexing tool for java problems. In: *CEUR Workshop Proceedings*. vol. 1009, pp. 60–63 (2013)
12. Ji, Z., Wei, Q., Xu, H.: Bert-based ranking for biomedical entity normalization. *AMIA Summits on Translational Science Proceedings* **2020**, 269 (2020)
13. Kumar, A.N.: Model-based reasoning for domain modeling in a web-based intelligent tutoring system to help students learn to debug c++ programs. In: *ITS*. pp. 792–801. Springer (2002)
14. Li, B., Zhou, H., He, J., Wang, M., Yang, Y., Li, L.: On the sentence embeddings from pre-trained language models. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pp. 9119–9130 (2020)
15. Miller, G.A.: Wordnet: a lexical database for english. *Communications of the ACM* **38**(11), 39–41 (1995)
16. Navarro, G.: A guided tour to approximate string matching. *ACM computing surveys (CSUR)* **33**(1), 31–88 (2001)
17. Olney, A.M.: Extraction of concept maps from textbooks for domain modeling. In: *International Conference on Intelligent Tutoring Systems*. pp. 390–392. Springer (2010)
18. Paul, D., Harvey, D.: *Java-how to program* (2012)
19. Recchia, G., Louwerse, M.M.: A comparison of string similarity measures for toponym matching. In: *The Comparatist : Journal of the Southern Comparative Literature Association* (2013)
20. Rivers, K., Harpstead, E., Koedinger, K.R.: Learning curve analysis for programming: Which concepts do students struggle with? In: *ICER*. vol. 16, pp. 143–151 (2016)

21. Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., Owens, B.B., Stephenson, C., Verno, A.: Csta k-12 computer science standards: Revised 2011. Tech. rep., New York, NY, USA (2011)
22. Tew, A.E., Guzdial, M.: Developing a validated assessment of fundamental cs1 concepts. In: Proceedings of the 41st ACM technical symposium on Computer science education. pp. 97-101 (2010)
23. Wang, S., Thompson, L., Iyyer, M.: Phrase-bert: Improved phrase embeddings from bert with an application to corpus exploration. In: EMNLP. pp. 10837-10851 (2021)
24. Wu, Z., Li, Z., Mitra, P., Giles, C.L.: Can back-of-the-book indexes be automatically created? In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. pp. 1745-1750 (2013)
25. Zechmeister, J.S., Zechmeister, E.B.: Introductory textbooks and psychology's core concepts. *Teaching of Psychology* **27**(1), 6-11 (2000)
26. Zhang, Y., Baldridge, J., He, L.: Paws: Paraphrase adversaries from word scrambling. arXiv preprint arXiv:1904.01130 (2019)