EasyChair Preprint

№ 775

FVT: A Fragmented Video Tutor for "Dubbing" Software Development Tutorials

Chunyin Nong, Qiao Zhang, Liguo Huang, Di Cui, Qinghua Zheng and Ting Liu

February 14, 2019

FVT: A Fragmented Video Tutor for "Dubbing" Software Development Tutorials

Chunyin Nong*, Qiao Zhang†, Liguo Huang†, Di Cui*, Qinghua Zheng*, Ting Liu*
*MOEKLINNS Lab, Department of Computer Science and Technology, Xi'an Jiaotong University, 710049, China
{xjtuncy20130818, cuidi}@stu.xjtu.edu.cn; {qhzheng, tingliu}@mail.xjtu.edu.cn
†Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX
{qiaoz, lghuang}@smu.edu

***Abstract*—Rapid growth of online resources provides massive supports for developers to fulfill their learning tasks. Text tutorial and video tutorial, as the two most common forms of online resources, may not be sufficient to meet developers' specific learning needs if used individually. Text tutorials are well-structured and easy to be navigated, however, digesting the text description may not be always pleasant. Video tutorials are intuitive and easy to follow, however, the pre-determined teaching flow can be very distracting if a specific piece of knowledge is targeted. In this study, we proposed a novel method and its supporting tool — Fragmented Video Tutor (FVT), to facilitate the learning tasks with specific objectives, aiming at augmenting the strengths of both forms of tutorials while offsetting the weaknesses inherent to use each form of tutorials by itself. Specifically, FVT leverages the code snippets extracted from video tutorials as the bridge to link the video fragments in a video tutorial to the relevant sections in text tutorials. The preliminary evaluation results demonstrate that the FVT is a feasible approach to link two forms of tutorials and improve the learning effectiveness and efficiency for developers.**

***Index Terms*—Software Education, Video Tutorial, Text Tutorial, Fragmented Video Tutor**

I. Introduction

Software developers are always required to quickly master new techniques and skills in a rapidly changing technology world [1]. In most cases, the learning objective can be very specific to a certain development task or problem resolution, such as how to use a specific API or deal with a certain type of exception, named as **targeted learning**. The learned techniques/skills might be put into practice immediately. However, the human learning is not only the mastery of knowledge points, but also the understanding of knowledge system. Hence, the goal of targeted learning is to learn a specific piece of new concept or technique in a short period of time and understand the big picture of its context.

Many efforts have been taken to enhance software development tutorials [2, 3, 4, 5], which could be classified into two categories: text tutorials and video tutorials. However, they may not be sufficient to meet the targeted learning needs if used individually. Text tutorials such as JavaDoc can be well-structured, comprehensive and logically organized into detailed and in-depth explanations. Although developers can easily search and locate the relevant materials in the text tutorial with the help of automated or semi-automated search tools, developers still need to go back and forth to read through the context, which can be verbose, and sometimes even a bit boring especially when the content is overwhelming or difficult to follow.

Video tutorials are usually pre-planned with step-by-step demonstration to quickly deliver a piece of coherent knowledge with visual cognition, which is usually appreciated by software developers who prefer visual and auditory cognition. For example, the YouTube video *Object-oriented Programming in 7 minutes*[1] gives learners the basic concepts of Object-oriented Programming in the shortest possible time and the easiest way. However, the lengthy videos also contain irrelevant contents which may distract developers from concentrating on the most relevant part. The way that video tutorials are created makes it difficult and time-consuming to navigate in order to search or review the relevant content in a set of videos if developers have specific and in-depth learning needs.

It would be an ideal for developers to achieve targeted learning if we can bridge the two forms of tutorials (text and video tutorials) to make them complementary to each other since the effectiveness and efficiency are keys to the success of targeted learning. In this paper, we propose a novel method for targeted learning and its supporting tool — Fragmented Video Tutor (FVT), by augmenting the strengths of both forms of tutorials while offsetting their inherent weaknesses. Leveraging the code snippets extracted from video tutorials as the bridge between text tutorial and video tutorial, FVT is able to quickly locate the relevant fragment in a set of video tutorials, which are linked to the corresponding sections in the text tutorials. An evaluation of the links generated by FVT is conducted with 177 video fragments extracted from two video tutorials with different language (50 in English and 127 in Chinese) and 400 sections of 38 chapters in a classic Java learning textbook. The performance of FVT shows the feasibility of using code snippets to link two forms of tutorials even with different spoken language in video tutorials. A usability study is also conducted with 10 students by asking them to answer a questionnaire once the assigned targeted learning tasks completed along with the FVT or text or video tutorial respectively. The main contributions of this paper include:

[1]https://www.youtube.com/watch?v=pTB0EiLXUC8

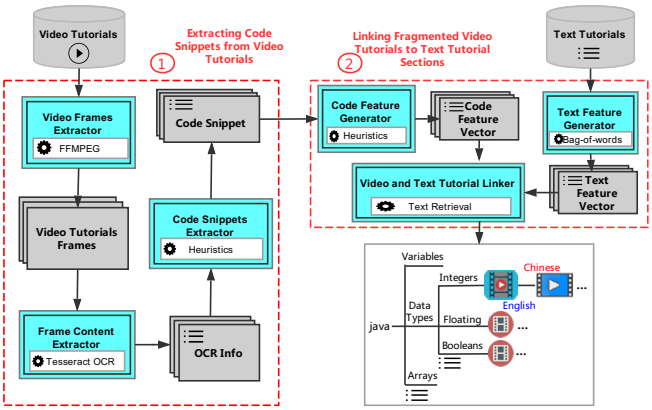

Fig. 1. The overview of FVT

- a novel method and tool to facilitate rapid targeted learning tasks in software development;
- demonstration of the feasibility to link text and video tutorial with code snippets.

The methodology is explained in Section II and followed by a case study in Section III. Section IV presents the evaluation setup and results. Section V summarizes the related work. Section VI concludes and envisages the future work.

II. Methodology

Fig. 1 shows an overview of the Fragmented Video Tutor (FVT) system. FVT facilitates targeted learning tasks in two steps: (1) extracting code snippets from the video tutorials and (2) linking the video tutorials to the sections in text tutorials with extracted code snippets. A well-organized and intuitive roadmap will be generated by FVT to guide learners from text tutorial to relevant video fragments or vice versa.

A. Extracting Code Snippets from Video Tutorials

As shown in the left box of Fig. 1, code snippets are extracted from video tutorials by 3 components: *Video Frames Extractor*, *Frame Content Extractor*, and *Code Snippet Extractor*. First, the *Video Frames Extractor* decomposes the video tutorials into a set of video frames with *FFmpeg*[2]. Second, the *Frame Content Extractor* captures the content of each frame with an optical character recognition (OCR) tool — *TESSERACT-OCR*[3], which returns the content of each video frame with a mixture of textual information and source code. *Code Snippets Extractor* with the island grammar [6] then extracts and parses the code snippets by differentiate source codes from natural language descriptions. Finally, the most representative code snippet is selected with the *MCIDIFF* [7], a state-of-the-art approach to compute differences across multiple instances of code clones.

B. Linking Fragmented Video Tutorials to Text Tutorial Sections

As shown in the upper right box of Fig. 1, in this step, the extracted code snippets in the last step will be used to link the video tutorial fragments to the relevant text tutorial sections. Firstly, the *Code Feature Generator* extracts a set of code features from code snippets including API sequence, API description, method name, and tokens [8] with 5 text retrieval (TR) techniques including TF-IDF [9], LSI [10], and LDA [11], as well as their combinations TF-IDF+LSI and TF-IDF+LDA on *gensim*[4]. Meanwhile, the *Text Feature Generator* extracts the word vectors from text tutorial sections, using the same TR techniques. Next, the *Video and Text Tutorial Linker* computes the cosine similarity for each pair of the code feature vector representing a video code snippet and the word vector representing a text tutorial section. We select no more than two text tutorial sections to link with each video fragment, whose similarity score are highest and also higher than the minimum limit. Finally, FVT can "dubbing" the text tutorials with the video, as shown in the bottom right box of Fig. 1.

III. Case Study

This section provides a case study to illustrate how FVT facilitates the targeted learning task. Suppose an Android developer who has the experience with Java would like to quickly learn a new language Kotlin for her career development. To make use of her breaks during work, she plans to prioritize the learning objectives and learn Kotlin incrementally. The first targeted learning task is to learn Null Safety in Kotlin first since the *NullPointerException* problem in Java has been notorious to all developers (a.k.a, the Billion Dollar Mistake[5]) while Kotlin's variable type system is aimed at eliminating the danger of null references from code.

As the most relevant text tutorial returned from Google search, the official document of Kotlin[6] provides a thorough explanation of null safety. However, the developer has to read through 16 code snippets to understand how to write the null safety code. Instead, an online video tutorial that claims the Kotlin can be learned after watching the one and half hour video would make this learning task more pleasant and efficient. To expedite the learning, she decides to first learn from the Kotlin video tutorial[7]. Nevertheless, the challenge is to search and locate the specific video fragment teaching null safety in the lengthy video tutorial. FVT recommends the relevant video fragment in the Kotlin video tutorial[7] that can be linked to the document of null safety of Kotlin[6] and save her effort to manually navigate and search through the video.

First, a set of code snippets is extracted by FVT from the video tutorial while each code snippet is associated with a location label which records the starting and ending time of the video fragment containing the code. As shown in the first transition on the left of Fig. 2, a code snippet can be extracted based on the video frame (as the screenshot) at 5,157s in the video[7]. Since the code at 4,945s of the video tutorial is completely different from the code after 4,947s and the Null

[2]https://www.ffmpeg.org
[3]https://github.com/tesseract-ocr
[4]https://radimrehurek.com/gensim/index.html
[5]https://qconlondon.com/london-2009/qconlondon.com/london-2009/speaker/Tony+Hoare.html
[6]https://kotlinlang.org/docs/reference/null-safety.html
[7]https://www.youtube.com/watch?v=H_oGi8uuDpA&t=4947s

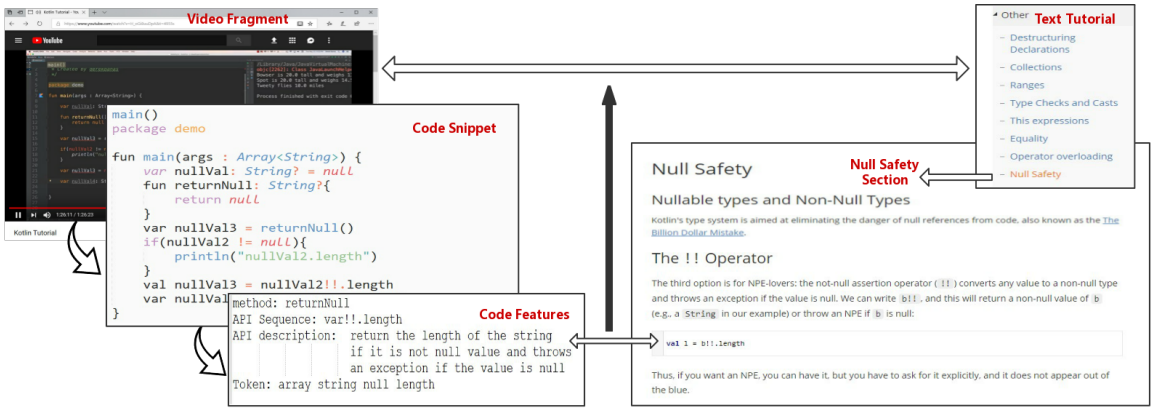

Fig. 2. A case study of using FVT to augment the learning of Null Safety in Kotlin

Safety is the last topic covered in the video, the extracted code snippet will be labeled with starting time of 4,947s and ending time of 5,183s. Then, the code features are extracted from the code snippets while the location label is inherited. In this case, a set of code features can be extracted including the method name ("returnNull"), API Sequence ("var!!.length"), API description ("return the length of the string if it is not null value and throws an exception if the value is null"), and Token ("array string null length")(shown in the second transition on the left of Fig. 2).

Next, FVT applies the 5 TR techniques to generate the code feature vector as mentioned in Section II-B. At the same time, the 5 TR techniques are applied to the Null Safety section in the text document to generate the word vector. Since the cosine similarity computed between the video code feature vector and the word vector generated from the section Null Safety in Kotlin[6] in the text tutorial is top ranked, FVT recommends that there is a link between the video fragment starting from 4947s in the original video tutorial[7] and the section in the text tutorial as shown in Fig. 2. From now on, she can use the linked video fragment to accomplish the learning of Null Safety in Kotlin while a cup of coffee is brewed instead of reading through the entire section on Null Safety in the text tutorial.

IV. Evaluation

Since the output of FVT is a set of links (roadmap) between video fragments extracted from a video tutorial and sections listed in a text tutorial, a preliminary evaluation is conducted to measure the relevance of generated links.

A. Datasets

In total, 177 video fragments (50 in English and 127 in Chinese) of Java programming language (JavaSE) tutorials are extracted respectively from YoutTube and atguigu[8], a Chinese software education institution. The length of video fragments ranges from 3 to 34 minutes with average 7 minutes in English videos and 9 minutes in Chinese videos. Since 10 out of 177 video fragments does not contain any code snippet, the remaining 167 video fragments are used as the input of FVT. We adopt "Java The Complete Reference Tenth Edition", a classical textbook for Java beginners, as the text tutorial. It is the official recommended textbook for Java by Oracle[9]. There are 400 sections of 38 chapters in the textbook to be linked with the relevant video fragments.

B. Manual Annotation

Two senior students manually and independently annotate the correctness of the links generated by FVT. Cohen-Kappa coefficient [12] is used to assess the level of agreement while 0.75 indicates a substantial level of agreement. Two students have to discuss and resolve the disagreement if the agreement is not substantial until the consensus is reached.

C. Metrics

The objective of FVT is to improve the effectiveness and efficiency of targeted learning in software development by providing learners the relevant video fragments if a specific topic described in a section of a text tutorial is identified as the learning objective. The relevant video fragments addressing the topic in the section can ease and expedite the learning. If no relevant video fragment is found to be linked to the text section, learners can still read the section in the text tutorial without the help from video tutorials. If an irrelevant video fragment is linked and recommended, learners will probably waste their time to watch it and eventually have to go back to square one. Hence, FVT shall focus on improving the relevance of the generated links. In other words, $precision$, which is the percentage of correct links returned by FVT, is more important for system users to measure the relevance of links generated by FVT.

D. Results

We have compared 5 TR techniques (TF-IDF, LSI, LDA, TF-IDF+LSI, and TF-IDF+LDA) to generate links between video fragments and text tutorial section. Table I shows the $precision$ of generated links by FVT employing the 5 linking

[8]http://www.atguigu.com/

[9]https://www.oracle.com/technetwork/topics/newtojava/documentation/index.html

TABLE I
PRECISION OF LINKS GENERATED WITH DIFFERENT TR TECHNIQUES

	TR Technique				
	TF-IDF	LSI	LDA	TF-IDF+LSI	TF-IDF+LDA
English	0.583	0.542	0.458	0.625	0.604
Chinese	0.542	0.58	0.475	0.609	0.634

strategies. The results demonstrate the feasibility of employing TR techniques in linking video fragments and text tutorial section.

Among the 5 techniques, TF-IDF+LDA achieves the highest $precision$ of 0.634 in the dataset with Chinese videos and 0.604 in the dataset with English videos, with an increase of up to 33.5% (0.634 vs. 0.475) as compared with any of its component techniques (LDA or TF-IDF). It shows that the combined TR techniques can significantly improve the relevance of the generated links as compared with the TR techniques applied individually. In other words, it is promising to improve the performance of FVT if one or more TR techniques can be combined to complement one another.

E. Error Analysis

An error analysis of incorrect links generated by FVT is also conducted to identify root causes. We found that most incorrect links occur when advanced Java programming techniques are focused, such as Java reflection or Java annotation. A possible reason is that the code snippets used in video tutorials for these topics are commonly designed along with a specific application scenario for better understanding (e.g., a class with application-specific method and interface names). It would be difficult to derive code features that can represent the learning subject when there are too many "noises". As the solution, we can consider extracting features from the custom class and method. For video tutorials that are aimed at teaching trivial and fundamental knowledge, such as the usage of different variable types or how to write a loop, FVT is very likely to link such video fragments to text sections incorrectly since the extracted code features are very general and common to a variety of topics throughout the entire text tutorial. It is possible to solve such problem by adding the language specified keywords or operators as the code feature. At last, if the video tutorial is about a specific type of algorithm (e.g., Bubble Sort, Dynamic Programming, etc.), the syntax features of code (e.g., nested loops, recursive call, etc.) are not considered in FVT at the current stage.

F. Usability Study

We also conducted a usability study to evaluate the proposed methodology with 10 junior undergraduate students. Among 10 students, 3 of them have 1 year Java programming experience and the rest 7 students have 3 years experience. We prepared both text and video tutorials that contain relevant contents of 5 different concepts to be the candidate targeted learning tasks. For each student, he/she was required to randomly select 3 out of 5 concepts to be learned with text tutorial only, video tutorial only, and both forms with FVT respectively. A questionnaire will be answered by each student to describe and rate his/her learning experience (advantage and disadvantage). As a result, 7 out of 10 students preferred FVT as a supportive tool to facilitate the targeted learning while 2 students choose text tutorial and 1 student choose video tutorial. The average rating (range between 1 to 5) are 3.2 for text tutorial, 3.3 for video tutorial, and 4.1 for both forms with FVT. We think FVT can help developers to better achieve targeted learning than using text or video tutorial individually.

V. RELATED WORK

Software video tutorials have been studied as a multimedia source to support developers [4, 13]. A recent study by MacLeod et al. [13] on programming videos on YouTube shows that video tutorials are effective in providing an introduction to a technology and demonstrating how a piece of software can be developed within an IDE. Yadid et al. [3] present ACE, a tool that combines language models and image processing techniques to extract source code from software development videos. Ott et al. [2] further improved the identification of code fragments using deep learning algorithm. Ponzanelli et al. [4, 14] employed the extracted code as features to segment development videos, which can efficiently assist developers to focus on the key point in video. Escobar et al. [1, 5] employed the transcripts in combination with other metadata (i.e., title and description) as features to automatically tag development videos. Comparing to abovementioned works, our methodology aims at linking video tutorials and text tutorials with the extracted code snippets from video. The link between two forms of tutorials is able to augment the learning experience for software developers.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel method and its supporting tool, Fragmented Video Tutor (FVT), to expedite the targeted learning for software developers. It augments the advantages of both text and video tutorials while offsetting the disadvantages of each form if used by itself. Specifically, FVT links the video fragments in video tutorial with the relevant content in text tutorials with 5 TR techniques and the code snippets extracted from video. The evaluation results show the feasibility of FVT to recommend links between two forms of tutorials with the $precision$ up to 0.634. To further improve the $precision$ of generated links, we will conduct experiments on additional code and text features as well as other TR and code search techniques. We will also conduct experiments on actual utility in supporting learning or programming effectiveness. Moreover, FVT will incorporate more application scenarios to adopt additional learning needs in industrial practices.

VII. ACKNOWLEDGMENT

This work was supported by National Key R&D Program of China (2016YFB1000903), National Natural Science Foundation of China (61632015, 61772408, U1766215, 61721002, 61532015, 61833015), Ministry of Education Innovation Research Team (IRT_17R86).

References

[1] E. Parra, J. Escobar-Avila, and S. Haiduc, "Automatic tag recommendation for software development video tutorials," in *Proceedings of the 26th Conference on Program Comprehension*. ACM, 2018, pp. 222–232.

[2] J. Ott, A. Atchison, P. Harnack, A. Bergh, and E. Linstead, "A deep learning approach to identifying source code in images and video," 2018.

[3] S. Yadid and E. Yahav, "Extracting code from programming tutorial videos," in *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. ACM, 2016, pp. 98–111.

[4] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza, "Too long; didn't watch!: extracting relevant fragments from software development video tutorials," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 261–272.

[5] J. Escobar-Avila, E. Parra, and S. Haiduc, "Text retrieval-based tagging of software engineering video tutorials," in *Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference on*. IEEE, 2017, pp. 341–343.

[6] A. Bacchelli, A. Cleve, M. Lanza, and A. Mocci, "Extracting structured data from natural language documents with island parsing," *automated software engineering*, pp. 476–479, 2011.

[7] Y. Lin, Z. Xing, Y. Xue, Y. Liu, X. Peng, J. Sun, and W. Zhao, *Detecting differences across multiple instances of code clones*. ACM, 2014.

[8] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 933–944.

[9] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[10] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse processes*, vol. 25, no. 2-3, pp. 259–284, 1998.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[12] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[13] L. MacLeod, M.-A. Storey, and A. Bergen, "Code, camera, action: how software developers document and share program knowledge using youtube," in *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*. IEEE Press, 2015, pp. 104–114.

[14] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, B. Russo, S. Haiduc, and M. Lanza, "Codetube: extracting relevant fragments from software development video tutorials," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 645–648.