Too Long; Didn't Watch! Extracting Relevant Fragments from Software Development Video Tutorials

Luca Ponzanelli[1], Gabriele Bavota[2], Andrea Mocci[1], Massimiliano Di Penta[3]
Rocco Oliveto[4], Mir Hasan[5], Barbara Russo[2], Sonia Haiduc[5], Michele Lanza[1]
[1] *Università della Svizzera Italiana (USI), Switzerland* — [2] *Free University of Bozen-Bolzano, Italy*
[3] *University of Sannio, Italy* — [4] *University of Molise, Italy* — [5] *Florida State University, USA*

ABSTRACT

When knowledgeable colleagues are not available, developers resort to offline and online resources, *e.g.,* tutorials, mailing lists, and Q&A websites. These, however, need to be found, read, and understood, which takes its toll in terms of time and mental energy. A more immediate and accessible resource are video tutorials found on the web, which in recent years have seen a steep increase in popularity. Nonetheless, videos are an intrinsically noisy data source, and finding the right piece of information might be even more cumbersome than using the previously mentioned resources.

We present CodeTube, an approach which mines video tutorials found on the web, and enables developers to query their contents. The video tutorials are split into coherent fragments, to return only fragments related to the query. These are complemented with information from additional sources, such as Stack Overflow discussions. The results of two studies to assess CodeTube indicate that video tutorials—if appropriately processed—represent a useful, yet still under-utilized source of information for software development.

CCS Concepts

•**Software and its engineering** → *Software maintenance tools;* Documentation;

Keywords

Recommender Systems, Mining Unstructured Data

1. INTRODUCTION

Developers need to continuously acquire new knowledge to keep up with their daily tasks. For example, to use a new library, learn a new programming language, or to develop mobile applications, they can use several resources to get the information they need. Especially online resources are on the rise [42], *e.g.,* forums, blogs, Question & Answer (Q&A) websites, slide presentations, due to the amount and diversity of available information.

ICSE '16, May 14-22, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2884781.2884824

When using search engines to find information for their tasks, developers often get a mix of results from these sources. Among them, a recent and rapidly emerging source of information are *video tutorials*. Video tutorials can be effective in providing a general and thorough introduction to a new technology, as they often include a step-by-step, learn-by-example introduction to how a technology should be applied in practice. A recent study by MacLeod *et al.* [19] investigated how and why software development video tutorials are created, and found that they share details such as software customization knowledge, personal development experiences, implementation approaches, application of design patterns or data structures. The study also highlighted key advantages of video tutorials compared to other resources, such as user manuals. These advantages include the ability to visually follow the changes made to the source code, to see the environment where the program is executed, to view the execution results and how they relate to the source code, and to understand a development activity in depth by looking at different levels of details. In essence, video tutorials can provide a learning perspective different and complementary to that offered by traditional, text-based sources of information.

Despite these benefits, *there is still limited support for helping developers to find the relevant information they require within video tutorials*. In many cases, video tutorials are lengthy, and lack an index to allow finding specific fragments of interest. Thus, to find information about a concept in a video tutorial, a developer can either watch the entire video, leading to effort and time wasted watching the irrelevant parts, or skim it, risking to miss important information.

Moreover, a developer may need information from diverse sources to thoroughly understand a new concept. For example, when learning to use a new library, a developer could benefit from an introductory video tutorial, complemented by discussions about known issues of that library from forums such as Stack Overflow. To the best of our knowledge, there is no support for integrating this kind of complementary, cross-platform information about a programming topic.

While approaches have been proposed to support developers by mining API documentation [35,36] and Q&A websites such as Stack Overflow [14,32], or by automatically synthesizing code examples from existing code bases [1,6,16,25], there is currently no approach aimed at leveraging relevant information found within fragments of video tutorials and linking these fragments to other relevant sources of information.

We propose CodeTube, a novel approach that effectively leverages the information found in video tutorials and other online resources, providing it to developers for a task at hand.

CodeTube recommends video tutorial fragments relevant to a given textual query, and complements them with Stack Overflow discussions related to the extracted video fragments. CodeTube starts from a set of videos relevant to a broad topic of interest (*e.g.,* Android development, J2EE). Then, CodeTube analyzes the videos and identifies when source code is being shown (*e.g.,* through the IDE) on the screen, by using a series of algorithms and heuristics aimed at identifying shapes and fragments of Java source code in the frame. Then, it isolates cohesive *video fragments*, *i.e.,* sequences of frames in which source code is being written, scrolled, or alternated with other informative material. The text contained in each video fragment is extracted and complemented with the text of the audio transcript occurring at the same time. Finally, all this information is indexed using information retrieval techniques. In addition, CodeTube searches and indexes Stack Overflow discussions relevant to each video fragment. A developer can then query CodeTube through a web interface, and obtain a ranked list of relevant video fragments with related Stack Overflow discussions.

CodeTube is currently available with a set of 4,747 indexed videos related to Android development extracted from YouTube. The videos currently considered resulted in a total of 38,783 fragments. The mean length of videos is 908s (1^{st} quartile 433s, median 684s, 3^{rd} quartile 1,073s); the fragments are one order of magnitude shorter, with a mean length of 66s (1^{st} quartile 52s, median 55s, 3^{rd} quartile 66s).

We evaluated CodeTube in two different studies. In the first study, 34 developers with Android experience performed an intrinsic evaluation of the results produced by CodeTube through an online survey. The participants evaluated (i) the coherence and conciseness of the video fragments produced by CodeTube, as well as their relevance to a query, as compared to the results returned by YouTube, and (ii) the relevance and complementarity of Stack Overflow discussions returned by CodeTube for specific video fragments. In the second study, we performed an extrinsic evaluation of the approach by introducing CodeTube to three leading developers involved in the development of Android apps. After that we asked them questions about the usefulness of CodeTube, focusing on the value of extracting fragments from video tutorials, and of providing recommendations by combining different sources of information.

Paper structure. Section 2 details CodeTube, while Section 3 and Section 4 describe and report the results of the intrinsic and the extrinsic evaluations. Threats to validity are discussed in Section 5. After a discussion of the related literature (Section 6), Section 7 concludes the paper.

2. CODETUBE OVERVIEW

CodeTube is a multi-source documentation miner to locate useful pieces of information for a given task at hand. The results are fragments of video tutorials relevant for a given textual query, augmented with additional information mined from other "classical", text-based online resources.

Figure 1 depicts the CodeTube pipeline. It is composed of (i) an offline analysis phase aimed at collecting and indexing video tutorials and other resources, and (ii) an online service where developers can search these processed resources. The analysis of video tutorials is currently limited to English videos dealing with the Java programming language. In the following we detail each step of the CodeTube pipeline.

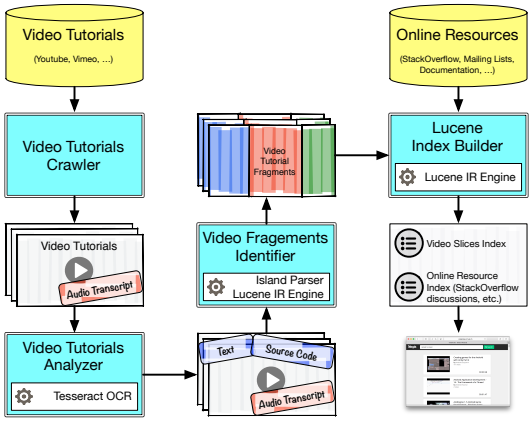


Figure 1: CodeTube: Analysis process.

2.1 Crawling and Analyzing Video Tutorials

The first step of the process is defining the topics of interest. The user provides (i) a set of queries Q describing the video tutorials she is interested in (*e.g.,* "Android development") and (ii) a set of related tags T to identify and index relevant Stack Overflow discussions (*e.g.,* "Android"). Each query in Q is run by the *Video Tutorials Crawler* using the YouTube Data API[1] to get the list of YouTube channels relevant to the given query $q_i \in Q$. For each channel the *Video Tutorials Crawler* retrieves the metadata (*e.g.,* video url, title, description) and the audio transcripts, which are either automatically generated or written by the author. Using Google2Srt[2] we extract the transcriptions for the videos. The crawling of video meta-information is performed on YouTube, but it can be extended to any video streaming service or video collection where the same type of meta-information and transcripts are available or can be extracted, *e.g.,* using a speech recognition API.

Once the videos have been crawled, their metadata is provided as input to the *Video Tutorial Analyzer*. It analyzes each video and extracts pieces of information to isolate video fragments related to a specific topic. The *Video Tutorial Analyzer* aims at characterizing each video frame with the text and the source code it contains. It uses multi-threading to concurrently analyze multiple batches of videos.

Frame Extraction. The analysis starts by downloading the video at the maximum available resolution. CodeTube uses the multimedia framework FFmpeg[3] to extract one frame per second, saving each frame in a `png` image. Given the set of frames in the video, we compare subsequent pairs of frames (f_i, f_{i+1}) to measure their dissimilarity in terms of their pixel matrices. If they differ by less than 10% we only keep the first frame in the data analysis since the two frames show *almost* the same information. This scenario is quite common in video tutorials where the image on the screen is fixed for some seconds while the tutor speaks. This optimization considerably reduces the computational cost of our process without losing important information. After obtaining the reduced set of frames to analyze, CodeTube performs the following *information extraction steps.*

[1] https://developers.google.com/youtube/v3/
[2] http://google2srt.sourceforge.net/en/
[3] http://www.ffmpeg.org/

Figure 2: Example frames from which CodeTube is able to extract code fragments.

English Terms Extraction. We use the tool tesseract-ocr[4] (Optical Character Recognition) to extract the text from the frame. We only consider correct English words by matching them with a vocabulary. OCR tools are usually designed to deal with text on white background (*i.e.,* paper documents). In order to cope with this, many OCR tools convert colored images to black and white before processing them. When using an OCR tool on video frames, the high variability of the background, and the potential low quality of a frame can result in a high amount of noise. Thus, after splitting composite words—based on camel case or other separators—we use a dictionary-based filtering, to ignore strings that are invalid English words[5].

Java Code Identification. In principle, the output of the OCR could be processed to extract the depicted Java constructs. However, such output often contains noise. Figure 2 shows three frames containing Java code. In frame 1 the code occupies the whole screen, and there is a clear background: the noise of the OCR output is limited. The noise increases in the Frames 2 and 3, due to the buttons, menu labels, the graphics on the t-shirt, *etc.* To limit the noise produced by the OCR we identify the sub-frame containing code using two heuristics, *shape detection* and *frame segmentation.*

Shape Detection. We use BoofCV[6] to apply shape detection on a frame, identifying all quadrilaterals by using the difference in contrast in the corners. This is typically successful to detect code editors in the IDE as in Frame 2.

Frame Segmentation. The shape detection phase could fail in identifying sub-frames with code. In Frame 3 of Figure 2 BoofCV fails because of missing quadrilaterals. In this case, we apply a segmentation heuristic by sampling small sub-images having height and width equal to 20% of the original frame size and we run the OCR on each sub-image. We mark all sub-images S_m containing at least one valid English word and/or Java keyword and we identify the part of the frame containing the source code as the quadrilateral delimited by the top-left sub-image (*i.e.,* the one having the minimum x and y coordinates) and the bottom-right sub-image (*i.e.,* the one having the maximum x and y coordinates) in S_m.

[4]https://github.com/tesseract-ocr
[5]We use the OS X English dictionary.
[6]http://boofcv.org/

Identifying Java Code. After identifying a candidate sub-frame, we run the OCR to obtain the raw text that likely represents code. Then, we use an island parser [3, 24] on the extracted text to cope with the noise, the imperfections of the OCR, and the incomplete code fragments. The island parser separates invalid code or natural language (water) from matching constructs (islands), and produces a Heterogenous Abstract Syntax Tree (H-AST) [33]. By traversing the H-AST we can exclude water nodes and keep complete constructs (*e.g.,* declarations, blocks, other statements) and incomplete fragments (*e.g.,* partial declarations, like methods without a body). If we are not able to match complete or incomplete Java constructs with any of the described heuristics, we assume that the frame does not contain source code.

2.2 Identifying Video Fragments

The *Video Fragments Identifier* detects cohesive fragments in a video tutorial using the previously collected information. We refer to Figure 3 to illustrate the performed steps. CodeTube starts by identifying video fragments characterized by the presence of a specific piece of code. The conjecture is that a frame containing a code snippet is coupled to the surrounding video frames showing (parts of) the same code.

Identifying the video frames containing a specific code snippet presents non-trivial challenges. First, a piece of code could be written incrementally during a video tutorial: if writing a Java class in a video tutorial lasts 3 minutes, all frames in the 3-minute interval will contain snippets of code related to that class and thus should be considered as part of the same video fragment. However, such code snippets are different (*i.e.,* they contain different programming constructs) due to the incremental writing. Second, the tutor could, to provide a line-by-line explanation, scroll the code snippet shown on video. Again, this causes frames showing the same code snippet to show different "portions" of it. Last, the tutor could interleave two frames showing the same snippet of code with slides or other material (*e.g.,* the Android emulator).

CodeTube overcomes these challenges and identifies video fragments characterized by the presence of a specific piece of code by comparing subsequent pairs of frames containing code to verify if they refer to the same code snippet. The frames depicted in red in Figure 3 represent "code frames", that is, frames containing code fragments. Given two code frames CodeTube verifies if they contain at least one common complete or incomplete Java construct. If so, the two frames are marked as containing the same code component. If not, we cannot exclude that the two frames do not refer to the same code; We have to take into account (i) possible imprecisions of the OCR when extracting the source code from the two frames, *i.e.,* it could happen that a Java construct is correctly extracted only in one of the two frames, and (ii) the possibility that a scrolling from one frame to another has hidden some constructs in one of the two frames.

If the island parser fails in matching a common construct in the two frames, we compute the Longest Common Substring (LCS) between the pixel matrices representing the code frames. Specifically, we represent matrices as strings, where each pixel is converted to a 8-bit grayscale representation. If the LCS between the two frames includes more than α of the pixels in the frames, CodeTube considers the two frames as showing the same code snippet.

The process adopted to tune the threshold α is reported in Section 2.3.

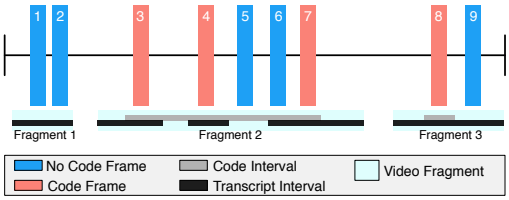


Figure 3: Identification of video fragments.

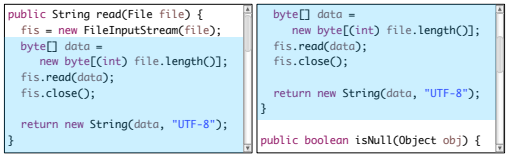


Figure 4: LCS between two frames showing the same code. The right frame is scrolled down by the tutor.

Note that the LCS is not affected by possible OCR imprecisions, and it does not suffer of problems related to the IDE scrolling, as shown in Figure 4 (in cyan the portion of the two frames identified as LCS). As a drawback, LCS is sensitive to zooming. Since the alignment of the proportions between two subsequent frames changes, LCS would fail in identifying a common part. Overall, given the advantages of the LCS over the Java constructs matching between the two frames via island parser, one may think that applying the LCS for each pair of code frames is the way to go. Unfortunately, the LCS is very expensive to compute due to the huge number of pixels composing a frame (a 1080p HD video has ∼2M pixels per frame), and estimating the LCS on each pair of code frames would require an unreasonable computation time.

For this reason, we adopt the LCS as a contingency strategy when the island parser is unable to identify common Java constructs in the two frames under analysis. To speed up the LCS computation we scale the frames to 25% of their size. In the example depicted in Figure 3, CodeTube compares the code frame pairs (3,4), (4,7), and (7,8), identifying the first two pairs as containing the same code snippet. As highlighted by the grey line below the frames, it identifies the first two cohesive "code intervals", *i.e.,* the first going from frame 3 to frame 7 and the second containing frame 8 only. The "non-code frames" 5 and 6 (blue in Figure 3) are included in the first code interval, since they are surrounded by two code frames (4 and 7) containing the same snippet.

In a subsequent step CodeTube analyzes the audio transcripts (black lines at the bottom of Figure 3) to refine the already identified code intervals (grey lines). CodeTube identifies the audio transcripts starting and/or ending inside each code interval. The audio transcripts are provided in the SubRip[7] format when extracted from YouTube's videos. In the example reported in Figure 3, three audio transcripts are considered relevant when refining the code interval going from frame 3 to 7. CodeTube uses the beginning of the first and the end of the last relevant audio transcript for a code interval to extend its duration and avoid that the code interval starts or ends with a broken sentence. The extended code interval represents an identified video fragment (Fragment 2—light cyan in Figure 3).

There might still be non-code frames in the video that have not been assigned to any video fragment (*e.g.,* frames 1 and 2 in Figure 3). These frames are grouped together on the basis of the audio transcript part they fall in. For example, the first two frames in Figure 3 are grouped in the same video fragment (Fragment 1), since they both fall in the same audio transcript part. As a final step, each subsequent pair of fragments is compared to remove very short video fragments and to merge semantically related fragments.

[7]https://en.wikipedia.org/wiki/SubRip

CodeTube merges together two subsequent fragments if one of two conditions applies:

1. Their textual similarity (computed using the Vector Space Model (VSM) [4]) is greater than a threshold β. Each video fragment is represented by the text contained in its audio transcripts and in its frames (as extracted by the OCR). The text is pre-processed by removing English stop words, splitting by underscore and camel case, and stemming with the Snowball stemmer[8].
2. One of the two fragments is shorter than γ seconds. This is done to remove short video fragments that unlikely represent a complete and meaningful fragment of a video tutorial.

2.3 Tuning of CodeTube Parameters

The performance of CodeTube depends on three parameters that need to be properly tuned:

- $\boldsymbol{\alpha}$ – minimum percentage of LCS overlap between two frames to consider them as containing the same code fragment;
- $\boldsymbol{\beta}$ – minimum textual similarity between two fragments to merge them in a single fragment;
- $\boldsymbol{\gamma}$ – minimum video fragment length.

To identify the most suitable configuration, one of the authors—who did not participate in the approach definition—built a "video fragment oracle" by manually partitioning a set of 10 video tutorials into cohesive video fragments. Then, we looked for the CodeTube parameters configuration best approximating the manually defined oracle. A challenge in this context is how to define the "closeness" of the automatically- and manually-generated video fragments.

Estimating Video Fragments Similarity. A video can be seen as a set of partitions (video fragments) of frames, where each frame belongs to only one partition, *i.e.,* the generated video fragments are clusters of frames. To compare the closeness of the video fragments generated by CodeTube and those manually defined in the oracle, we used the MoJo effectiveness Measure (MoJoFM) [43], a normalized variant of the MoJo distance, computed as:

$$MoJoFM(A,B) = 100 - \left(\frac{mno(A,B)}{max(mno(\forall E_A, B))} \times 100 \right)$$

where $mno(A,B)$ is the minimum number of *Move* or *Join* operations needed to transform a partition A into a partition B, and $max(mno(\forall\ E_A, B))$ is the maximum possible distance of any partition A from the partition B. Thus, $MoJoFM$ returns 0 if A is the farthest partition away from B, and returns 100 if A is exactly equal to B.

[8]http://snowball.tartarus.org

Table 1: Parameter tuning intervals.

Parameter	Min	Max	Δ
α	5%	50%	5%
β	10%	80%	5%
γ	1,000s	120,000s	10,000s

While MoJoFM is suitable to compare different partitions (video fragments) of the same elements (frames), we must take into account that video fragments are characterized by a constraint of sequentiality (*i.e.,* they can only contain subsequent frames). This could lead the MoJoFM to return high values (similarity) even when applied to two totally different video partitions. For example, consider the video frames $F = \{1, 2, 3, 4, 5, 6\}$ and two sets of video fragments where the first set, $A = \{1, 2, 3, 4, 5, 6\}$, contains a unique partition (video fragment) with all the elements (frames), and the second set, $B = \{\{1, 2, 3\}, \{4, 5, 6\}\}$, contains 2 partitions of size 3. Since the MoJoFM is not a symmetric function, it would return $MoJoFM\,(A, B) = 25.0$ and $MoJoFM\,(B, A) = 80.0$, *i.e.,* two different values, despite the fact that the two partitions are the same. Keeping a one-way comparison between the oracle and the obtained video fragments undermines the tuning phase. To avoid this, yet keeping a margin of approximation, both sides of the MoJoFM should be taken into account. Two sets of fragments will tend to have the same value if they are close in their partitioning. For this reason, we calculate the similarity in both directions and compute their mean value:

$$closeness(A, B) = \frac{MoJoFM(A, B) + MoJoFM(B, A)}{2}$$

In doing so, spikes of high values for the *MoJoFM* between two sets of video fragments for one direction are lowered or preserved depending on the opposite.

Estimating the Most Suitable Parameter Configuration. For each parameter, we identified a set of possible values. Table 1 shows the intervals we adopted, and the step (Δ) used whenever a new combination is generated. In total, we experimented 1,800 different parameter combinations, adopting the one with the top ranked MoJoFM (α=5%, β=15%, γ=50s) for the full-fledged analysis phase.

2.4 Integrating Other Sources of Information

CodeTube can be enriched by mining other online resources, as our long-term goal [29] is to offer a *holistic* point of view on the information at disposal, also because we argue that no single type of resource can offer exhaustive assistance. To illustrate this, we added as an additional online information source the Stack Overflow data dump. We mined and extracted discussions related to the topics of the extracted video tutorials, pre-processed them to reduce the noise, and made them available to CodeTube.

The last step in the data pre-processing of CodeTube consists in indexing both the extracted video fragments and the Stack Overflow discussions, using Lucene[9], where each video fragment is considered as a document. For Stack Overflow we separately index each question and answer for each discussion. The text pre-processing phase is identical to the one explained in Section 2.2. The text indexed for a video fragment is represented by the terms contained in its frames and audio transcripts. The text indexed for the Stack Overflow post is represented by the terms they contain.

[9]https://lucene.apache.org/

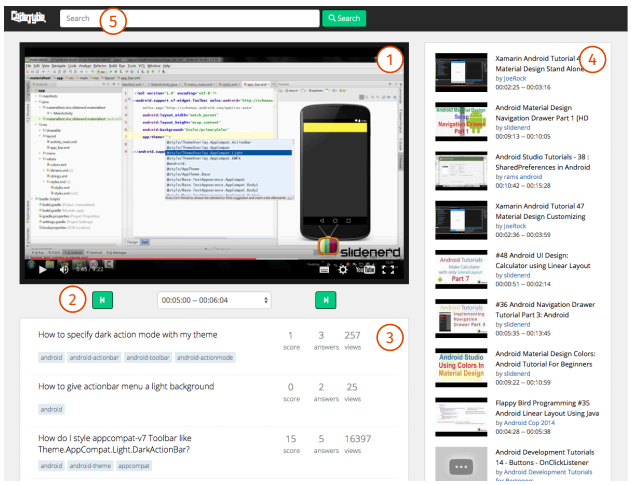


Figure 5: CodeTube: User interface.

2.5 The CodeTube User Interface

CodeTube provides a service that allows user to search, watch, and navigate the different fragments of a video. A detailed description of the CodeTube's service is reported in a companion tool demonstration paper [**?**].

Figure 5 shows the user interface of CodeTube. When a video fragment is selected for watching from the search results, CodeTube uses the YouTube player (1) provided by the YouTube API[10]. The video starts at the time devised by the selected fragment. CodeTube provides an additional controller (2) to visualize the timestamps of the fragments identified by our approach, select a specific fragment, or move to the next/previous fragment. During the video playback, the selector underneath the video player keeps the pace of the video timing and shows the current fragment. When a new fragment is reached, or the user jumps to it, CodeTube automatically extracts a query from the text contained in the fragment (*i.e.,* transcripts and OCR output of the frames it contains), queries both the index of Stack Overflow and of the video fragments, and updates the related discussions (3) and the suggested YouTube video fragments (4). A search bar (5) is always available to the user to run new queries.

3. STUDY I: INTRINSIC EVALUATION

The *goal* of this study is to evaluate CodeTube with the *purpose* of determining the quality of the extracted video fragments and related Stack Overflow discussions perceived by developers. The *quality focus* concerns (i) the perceived benefits and obstacles in using video tutorials during development and (ii) the quality of video fragments (cohesiveness, self-containment, relevance to a query) and Stack Overflow discussions (relevance and complementarity to the video fragment) mined by CodeTube. The four research questions (RQ) the study aims to answer are:

RQ$_1$: *What are the perceived benefits and obstacles of using video tutorials?*

[10]https://developers.google.com/youtube/js_api_reference

RQ$_2$: *To what extent are the extracted video tutorial fragments cohesive and self-contained?*

RQ$_3$: *To what extent are the Stack Overflow discussions identified by CodeTube relevant and complementary to the linked video fragments?*

RQ$_4$: *To what extent is CodeTube able to return results relevant to a textual query?*

The *context* of the study consists of *participants* and *objects*. The *participants* have been identified using convenience sampling among personal contacts of the authors, and by sending invitations over mailing lists for open-source developers. In total, 40 participants completed the survey. The *objects* of the study are the set of 4,747 video tutorials about Android development indexed in CodeTube. From these video tutorials, CodeTube extracted a total of 38,783 fragments.

3.1 Study design and procedure

The study has been conducted using an online survey questionnaire, through which we asked questions to the potential respondents to assess the results of CodeTube. The survey questionnaire is composed of three sections, preceded by preliminary assessment of the primarily activity (industrial/open source developer, student, academic), programming experience, and specific experience about Android development of respondents. The first section (addressing **RQ$_1$**) contains questions having an exploratory nature and aimed at understanding (i) how often and in which circumstances respondents use video tutorials and Q&A Websites, (ii) whether they found useful information there, and (iii) how they react to video tutorials being too long (*e.g.,* scroll it, watch it anyway, or give up). We also asked participants what the main points of strength and weakness of video tutorials are, compared to standard documentation and Q&A Websites.

The second section shows to respondents three video fragments extracted by CodeTube, as well as the original video tutorial from YouTube. Then, it asks (**RQ$_2$**) whether the fragment is cohesive and self-contained. For each video fragment, we also show the top-three relevant Stack Overflow posts, and ask (**RQ$_3$**) to what extent they are relevant and complementary to the video tutorial fragments. While approaches to recommend Stack Overflow discussions exist [32], our aim is to determine whether the textual content of the video tutorial fragment can be used to retrieve relevant discussions. For each respondent, the second section is repeated for two video tutorials randomly chosen from a sample of 20 video tutorials randomly selected from the 4,747.

The third section aims to assess the relevance of the top-three returned video fragments to a given query (**RQ$_4$**). As a baseline for comparison, we evaluate the relevance of the top-three videos returned by YouTube using the same query. The query shown to each respondent is sampled from a set of 10 queries formulated by graduate students at Florida State University, having a long experience in Android development. The queries are related to typical Android problems, *e.g.,* sending logs to servers, initiate activities in background, animate transitions, access accelerometer data, stopping background services, or modifying the UI layout.

The queries are generic, and YouTube is likely able to return as relevant results as CodeTube. Only specific queries, referring to code elements—not contained in YouTube metadata—would show the advanced of the indexing capabilities of CodeTube. Instead, we are interested in showing that, for the typical queries a developer formulates, CodeTube returns at least as relevant as YouTube, but consisting in shorter, cohesive and self-contained fragments.

Finally, after the third section, we asked the respondents to evaluate, through an open comment, the main points of strength and weakness of CodeTube. All assessment-related questions follow a 3-level Likert scale [27], *e.g.,* "very cohesive", "somewhat cohesive", and "not cohesive". We limit the number of video fragments, Q&A discussions and queries for each respondent to avoid the questionnaire being too long. Before sending the questionnaire to perspective respondents, we ran a pilot study to assess its estimated duration, which resulted to be between 25 and 40 minutes.

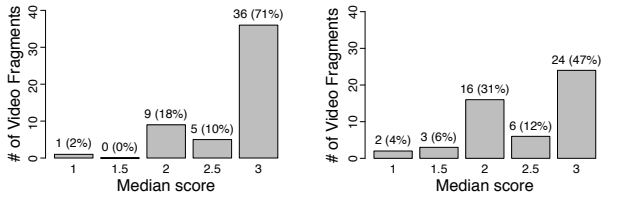
The questionnaire was then uploaded on the Qualtrics[11] online survey platform, and a link to the questionnaire was sent via email to the invitees. We made it clear that anonymity of participants was presented and data were only published in aggregate form. The *Qualtrics* survey platform allowed us to achieve randomization and balancing, by automatically selecting video tutorials (with related Stack Overflow discussion) and queries to be evaluated by each respondent. After sending out the invitation, invitees had two weeks to respond.

3.2 Study results

Out of the 40 study participants, 6 declared to have no experience in Android development. Since the video tutorials considered in the study were not introductory but related to specific Android topics, we excluded their answers. Excluding these, we collected a total of 180 video tutorial fragment evaluations (with respect to their cohesiveness and self-containment), 540 SO discussion evaluations, and 90 video tutorial fragment evaluations with respect to a query. Ideally, we could have collected more evaluations, but we have to consider that each of them requires respondents to watch a video tutorial fragment (and in the case of the queries also the whole video tutorial itself), hence we had to be realistic in the workload required by the targeted respondents. With such numbers and given our design, each fragment and SO discussion received a number of evaluations varying between 3 and 5, except for 3 videos and 2 queries, that, due to the exclusion of some participants motivated above, received less than 3 evaluations. These videos and queries were excluded from the analysis. With a set of videos smaller than our 20 we could have obtained more responses per fragment and SO discussions. We decided to favor the evaluation of a relatively larger set—and variety, hence more generalizability—of videos rather than having more responses and therefore more reliable evaluation for each video.

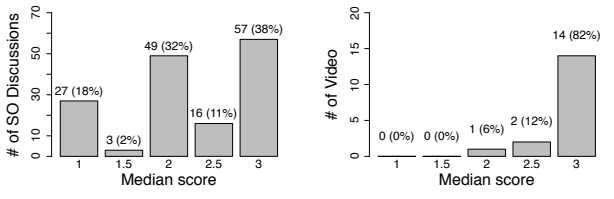
The population who completed our survey is composed of 70.6% of professional and open source developers, 17.6% of master students, and 11.8% of PhD students. The majority of developers in the population guarantees, on average, a higher level of experience: 32.3% of the population has more than 10 year experience, 17.5% has between 5 and 10 years, 38.3% between 3 and 5 years, 11.8% between 1 and 3 years. No one declared less than 1 year of programming experience. When asked about Android programming experience, the majority (38.3%) declared less than 1 year of experience, followed up by 23.5% of respondents with more than 3 years experience, 20.5% between 2 and 3 years, and 17.6% between 1 and 2 years of experience.

[11] https://az1.qualtrics.com



(a) Cohesion (b) Self-Containment

Figure 6: Distribution of median cohesion and self-containment scores for the assessed video fragments.



(a) Relevance (b) Complementariness

Figure 7: Relevance of Stack Overflow discussions to video fragments, and complementariness to videos.

3.2.1 RQ_1: What are the perceived benefits and obstacles of using video tutorials?

The usage of video tutorials either happens on a weekly (38.2%) or monthly (35.3%) basis. 3% declared to use video tutorials on a daily basis; nobody declared to never use them. Video tutorials are unlikely to help bug/error fixing (5%), but are the primary means to learn new concepts (43%).

When asked to provide open comments on the weaknesses and strengths of video tutorials, respondents pointed out different key aspects. The primary point of strength is the step-by-step nature of a video. One respondent wrote *"As opposed to Q&A Websites, video tutorials describe a complete process step-by-step. The visualized flow of actions is particularly useful in setting up working environments"*, another emphasized the *"possibility to see the complete interaction of the developer with the IDE"* and *"how a specific library is imported before it is used in the code. This does not hold when you simply copy and paste code from Websites"*. Another point of strength identified by respondents concerns the guidance given by a tutor. One respondent reported that *"there is a 'real' person talking with you, so it is easy to learn new concepts"*, while another respondent emphasized the fact that *"you can see what the tutor does"*.

The primary weakness identified by respondents concerns time. When a video tutorial is too long, respondents said they would either try to scroll it to seek the relevant information (47%), or give up to find alternative sources (53%). Nobody opted for the third option, *i.e.,* watching the whole video anyway. Respondents generally consider videos too long and slow and not suited *"if you need to quickly solve a problem"*, or *"if you need just a small piece of information"*. One of the respondents reported how *"due to time constraints during software development I cannot always watch the entire tutorial"*. The lack of searching and indexing functionalities of the contents of a video is also considered a weakness. One of the respondents claimed that *"browsing is not easy, unless the video has an index to navigate through the concepts/sections in the video"*, while another highlighted how *"searching for a particular piece of information in the whole video is much harder than doing the same in a text document"*.

3.2.2 RQ_2: To what extent are the extracted video tutorial fragments cohesive and self-contained?

Figure 6(a) shows the distribution of median perceived cohesiveness scores for the 51 fragments of the 17 videos that received at least three evaluations. The first quartile, median and third quartile of the distribution are 2, 3, and 3, respectively. A large majority (71%) of the evaluated fragments achieved a score of 3 (cohesive), and only one fragment was considered as not cohesive.

Figure 6(b) shows the distribution of the median self-containment score of the video fragments as provided by the evaluators. In this case, the first quartile, median, and third quartile are 2, 2.5 and 3, respectively. As one can notice from the figure, the proportion of video fragments that received a median score of 3 is lower than for cohesiveness (47%). This is not surprising because obtaining self-contained fragments—and hence understandable without watching the rest of the video—is more challenging than achieving a high cohesiveness. Nevertheless, the achieved cohesiveness is overall more than reasonable as 59% of the fragments achieve a score greater than 2, and only 10% of them were considered as not self-contained (score less than 2).

Examples. Let us consider a video having high cohesion[12]: the tutor introduces the problem of implementing animations, and sets up the code for the next steps (fragment 1), implements the actual core of the animation (fragment 2), and runs the code to verify the animation (fragment 3). In almost all the cases the fragments identified by CodeTube resulted to be very coherent. A counter-example lies in the last fragment of another tutorial[13], that is considered not cohesive by the majority of the participants.

Let us now consider one positive case of self-containment[14]. In the first two fragments of the video, the tutor initializes the views, and adds the contents to a `TextView`. Participants perceived a positive self-containment of these fragments because they strongly depend on each other. Absolute positive consensus is reached on the last fragment, where the tutor runs and tests the sample application he implemented.

The last video[15] is a case of negative self-containment across all the identified fragments. This video is a particular corner case where the tutor gives for granted few basic notions and thus the video itself exhibits a lack of self-containment, thus making CodeTube not effective.

3.2.3 RQ_3: To what extent are the Stack Overflow discussions identified by CodeTube relevant & complementary to the linked video fragments?

Figure 7(a) shows the distribution of the median perceived relevance of the Stack Overflow discussions associated to the video fragments of each video tutorial considered in the study.

[12]http://codetube.inf.usi.ch/view?id=BVhBEdLVxGY
[13]http://codetube.inf.usi.ch/view?id=1C4RNtaPXxE
[14]http://codetube.inf.usi.ch/view?id=pNxsCXicqGM
[15]http://codetube.inf.usi.ch/view?id=ROA7n27JFNs

The distribution first quartile is 2, the median 2 and the third quartile 3. On the one hand, the perceived relevance is relatively low, with only 38% of the Stack Overflow discussions achieving a median relevance of 3.

On the other hand, if we look at Figure 7(b), we notice that the distribution is polarized towards the maximum value—first quartile, median and third quartile equal to 3—with 14 (82% of the total) of the videos where the Stack Overflow discussions were considered as complementary. Results indicate that, while respondents only considered the retrieved discussions fairly relevant to the fragments from where the queries were generated, they almost totally agreed about the complementarity of the provided information. We believe that video tutorials have a different purpose than Stack Overflow discussions. The former have an introductory, step-by-step guide to a given problem, the latter discuss a specific problem/answering a specific questions. For instance, given the fragments of a video[16] showing how to code a layout in XML for Android, CodeTube retrieved and suggested Stack Overflow discussions concerning typical problems a developer could encounter, like a button not showing up[17], or not well-formed XML[18].

3.2.4 RQ_4: To what extent is CodeTube able to return results relevant to a textual query?

In the last part of the survey, we asked participants to evaluate the top-three results that CodeTube and YouTube retrieved for a set of 10 queries. Each participant evaluated the relevance of a result with respect to the query by following a three-level Likert scale [27], *i.e.,* "very related", "somewhat related", and "not related". We use the Normalized Cumulative Discounted Gain (NDCG) [21] to aggregate the results.

Similarly to what done for the other research questions, queries with less than 3 replies are ignored. The NDCG is thus calculated on a set of 8 queries out of the initial 10. We obtained $NDCG_{CT}(Q,3) = 0.67$ and $NDCG_{YT}(Q,3) = 0.63$ for CodeTube and YouTube , respectively. Even if CodeTube seems to perform slightly better than YouTube, a statistical analysis of the $NDCG_{YT}$ and $NDCG_{CT}$ distributions, performed using the Wilcoxon paired test, did not show the presence of a statistically significant difference (p-value=0.49). Even though the data collected is not enough to draft any statistically significant conclusion, there are some considerations to make. First, when extracting the top-three results from YouTube we removed all the retrieved videos that are not included in the CodeTube dataset. This makes the comparison unfair for our approach. Second, YouTube recommends entire videos, while CodeTube recommends specific fragments. Thus, our approach is potentially more focused even if both the fragment and the whole video recommended by YouTube are equally relevant.

3.2.5 CodeTube: Strengths and weaknesses

In the last part of the questionnaire we asked participants to freely comment about CodeTube. The participants have in general a positive impression of CodeTube. The UI has been appreciated by some of the participants. For example, one of the respondents reported *"CodeTube looks very useful, added to the bookmarks!"*, while another wrote *"excellent work, [..] the idea behind CodeTube is brilliant"*. The extraction of fragments from video tutorials has been appreciated and considered *"very useful for developers who are already knowledgeable about the topic, they can save a lot of time"*.

The possibility of having complementary sources of information, *e.g.,* Stack Overflow has been appreciated by some participants. One of them reported that *"the concept is amazing, and has a lot of possibility of improvement, given the huge amount of different sources of data available"*, while other participants asked for additional features to improve this functionality. One participant asked for *"the possibility to search for SO discussions directly below the video"*, while another wondered that *"it would be nice if the tool can provide a summary/description that describes the context"*.

[16]http://codetube.inf.usi.ch/view?id=1UE7dJ7DK
[17]http://stackoverflow.com/questions/16687060
[18]http://stackoverflow.com/questions/11829271

4. STUDY II: EXTRINSIC EVALUATION

A successful technological transfer is the main target objective for each prototype tool. Thus, the *goal* of this second study is to extrinsically investigate CodeTube's industrial applicability. Specifically, the research question we aim to answer with this second evaluation is:

RQ$_5$: *Would* CodeTube *be useful for practitioners?*

The *context* of the study is represented by three leading developers—all with more than five years of experience in app development—of three Italian software companies, namely Next, IdeaSoftware, and Genialapps.

4.1 Study design and procedure

We conducted semi-structured interviews to get quantitative and qualitative feedback on CodeTube. Each interview lasted two hours. During the interview we let developers explore CodeTube for about 90 minutes, searching for video tutorials on specific technology or to fix problems. Each interview was based on the think-aloud strategy. We also explicitly asked the following questions: (1) Do you use video tutorials during development tasks? (2) Would the extraction of shorter fragments make you more productive? (3) Is the multi-source nature of CodeTube useful? (4) Are you willing to use CodeTube in your company?

Participants answered each question using a 4-point Likert scale: absolutely no, no, yes, absolutely yes. The interviews were conducted by one of the authors, who annotated the answers as well as additional insights about the strengths and weaknesses of CodeTube that emerged during the interviews.

4.2 Study results

Nicola Noviello, Project Manager @ Next. Nicola positively answered to our first three questions (*i.e.,* "absolutely yes"). Nicola declared to use video tutorials daily; "*they are particularly useful for senior and junior developers for both learning a new technology or finding the solution to a given problem. I see very often my developers on specialized YouTube channels searching for and watching video tutorials.*" Nicola also appreciated the multi-source nature of CodeTube; "*the video tutorial provides the general idea on the technology, while Stack Overflow discussions are particularly useful to manage alternative usage scenarios and specific issues.*". Regarding the extraction of fragments, Nicola commented that "*I usually discard video tutorials that are too long, because when I try to scroll/fast forward it to manually locate segments of interest, I am generally not able to find*

what I need. I strongly believe that the relevant segment is there but randomly scrolling a video tutorial is not worthwhile! I prefer to look for more focused video tutorials.". Nicola then confirmed that the availability of shorter fragments would make him much more productive.

Nicola answered "yes" to the question related to the usefulness of CodeTube; "*I did not answer absolutely yes because of the limited number of indexed tutorials. However, I strongly believe that the tool has an enormous potential.*". Nicola declared that he will present the tool to a newcomer trainee to quantify to what extent the tool is useful for developers that have a little knowledge on the Android world; *"I usually suggest to trainees to look for and watch video tutorials but very often they are not able to find the right information. I would like to see whether* CodeTube *is able to mitigate such a problem.*".

Luciano Cutone, Project Manager @ IdeaSoftware. Luciano positively answered to our first three questions; "*I love video tutorials but several times they are too long and I do not have enough time to watch whole videos. Thus, I have to scroll the video hoping to identify relevant segments. This takes time and makes video tutorials less effective. With* CodeTube *life will be easier!*" When exploiting different sources of information, Luciano works differently from Nicola; "*I like the idea of having video tutorials together with Stack Overflow discussions. However, the main source of information for me is Stack Overflow, while video tutorials should be used to fix problems; if I need to apply a new technology, I would like to start from Stack Overflow since there I can find snippets of code that I can copy and paste into my application. Then, if something goes wrong, I try to find a video tutorial to fix the problem.*". Luciano also suggested a nice improvement; "*Besides the integration of video tutorials with discussions on forums, I suggest to add another source of information, namely sample projects. Specifically, on GitHub there are several sample projects that explain how to apply specific technologies. Having them together with video tutorials and Stack Overflow discussions would be fantastic.*" Another suggestion was the addition of a voting mechanism to provide information on the usefulness and the effectiveness of a specific (fragment of a) video tutorial. Luciano answered "absolutely yes" to our last question (*i.e.,* the one related to the usefulness of CodeTube); "*I just added* CodeTube *to my bookmarks. This is the tool I wanted. I spent several hours of the day and of the night on YouTube and Stack Overflow to fix problems or learn new things. This is part of my job, unfortunately. With* CodeTube *I am sure that I will find relevant information quickly. I can finally go back to sleep during the night!*". The day after the interview, we got a text message from Luciano: "*I have just used* CodeTube *this morning. I was looking for something related to Android WebSocket. I found all I needed. Awesome!*".

Giuseppe Socci, Project Manager @ Genialapps. Giuseppe answered "absolutely yes" to our first question, stating that in his opinion "*Video tutorials are a crucial source of information for learning a new technology*". Instead, he answered "no" to our second research question related to the extraction of fragments; "*I am not 100% sure that extracting shorter fragments makes you more productive. It depends on the scenario where the video tutorial is used. To me, video tutorials should be used to learn a new technology. In this case I should watch the whole video. However, there could be cases where you just need to fix a problem or have some clarifications on a specific part of the technology. In this case watching fragments instead of whole videos could be worthwhile*".

Giuseppe suggested a way to make the tool more usable based on his way of interpreting video tutorials; "*the search of a video tutorial should be scenario-sensitive. Before searching, the user should specify why she is searching for a video tutorial. The first option could be 'I have a problem'. In this case, the search is based on fragments. The second option could be 'I want to learn'. Here, whole videos should be retrieved*". As well as the other two developers, Giuseppe liked the integration of video tutorials with forum discussion (he answered "absolutely yes" to our third research question). Consistently with findings of Study I (Section 3.2.5), he highlighted the need for manually refining queries when retrieving Stack Overflow discussions: "*all the visualized Stack Overflow discussions are related to a specific video tutorial. However, Stack Overflow discussions should be useful to resolve a problem I encountered when applying the technology explained in the video tutorial. Thus, it might be useful to filter the retrieved discussion by a specific query (e.g., the type of error I got).*". Finally, Giuseppe answered "yes" to our final question; "*I think that the tool is nice. You are trying to solve an important and challenging problem, that is merging accurately different sources of information in order to make them more productive.*". Giuseppe also gave a suggestion on how to improve the visualization of the relevant fragments; "*After submitting a query,* CodeTube *provides the list of relevant video fragments. However, it is quite difficult from the title of the video and the cover image to identify the most relevant one. I strongly suggest to show for each video the relevant textual part of the video content, similar to the part of the text in a Web page content visualized by Web search engines. The same approach could be used also to make the navigation of the fragments of a specific video easier.*".

4.3 Reflection: Approach vs. Tool

The reception of CodeTube was positive. All leading developers saw great value and even greater potential in this line of work. Several improvement suggestions, obtained also in Study I, regard the tool that embodies our approach, which we are currently considering. Clearly, tools can always be improved, given sufficient time and human resources. However, we would like to emphasize that, stepping beyond mere implementation and UI concerns, the main contribution of the paper lies in the underlying approach.

5. THREATS TO VALIDITY

Threats to *construct validity* are mainly related to the measurements performed in our studies, and Study I in particular. Instead of using proxy measures, we preferred to let developers evaluate video fragments and their related Stack Overflow discussions. Subjectiveness of such an evaluation was mitigated by involving multiple evaluators for each video, although as explained in Section 3.1 we favored the number of videos over the number of responses per fragment.

Threats to *internal validity* concern factors internal to our studies that could have influenced our results. One possible problem is that the evaluation could have been influenced by the knowledge of respondents about the topic. We mit-

igated this threat by discarding responses of participants not having any knowledge about Android. In addition, the evaluation is mainly related to cohesiveness, self-containment and relevance of video fragments, and relevance and complementariness of Stack Overflow discussions, rather than to how they would be helpful for the respondents.

In such cases, the bias represented by respondents is fairly limited. The videos used in the survey have been randomly sampled by considering 7 minutes as maximum video duration, and three as maximum number of fragments for each video and query results. These limitations have been introduced to restrict the survey duration to a reasonable time.

Threats to *external validity* concern the generalizability of our findings. Our evaluation is intendedly limited to video tutorials related to Android development, though further evaluation with a wide variety of tutorials is desirable. Nevertheless, such tutorials are not much different—in their structure and content—than other Java development tutorials. Both CodeTube and its evaluation need to be extended in the future to support multiple languages and, possibly, to evaluate it when processing tutorials involving multiple languages and pieces of technology. Finally, the validity of the second study is limited to the three very specific mobile app development contexts considered.

6. RELATED WORK

To our knowledge, the only work investigating the use of video tutorials by developers is the study by MacLeod *et al.* [19] which, as discussed in the introduction, represents the underlying motivation behind CodeTube. We discuss related work about (i) recommender systems for software documentation and code examples, (i) multimedia retrieval and processing, and (iii) use of multimedia in learning.

Recommender systems for software documentation and code examples. Numerous approaches have been proposed to provide developers with official or informal documentation for their task at hand, as well as code samples they can reuse. Among the various informal documentation sources, Stack Overflow has been used by many recommender systems [7, 30–32, 35, 40]. Other recommenders have focused on recovering links between code and documentation [2], with some focusing on recommending or enhancing API documentation [36, 39]. Among these, the work of Petrosyan *et al.* [28] is the most related, as it analyzed tutorials to extract fragments explaining API types. With respect to such approaches, CodeTube is specific for analyzing video tutorials and recommending their cohesive and self-contained fragments. Other approaches aimed to retrieve code elements or code examples relevant to a task at hand from the current project or its code base [1, 6, 8, 16, 17, 20, 25], or from online resources [5, 12, 13, 15, 34, 37–39, 41]. Other work suggested relevant web discussions to help solve coding problems [18].

The infrastructure of CodeTube is designed such that any source of documentation can potentially be used to complement the information extracted from video tutorials.

Multimedia Processing and Retrieval. Multimedia information retrieval focuses on extracting and retrieving relevant information from multimedia resources (*e.g.,* images, audio, or video). One problem in the field is splitting a video into semantically coherent fragments. Existing approaches usually employ supervised machine learning techniques applied to various textual, acoustic, and visual features [9] to resolve such an issue. Galuščáková and Pecina [11] explored the use of Passage Retrieval segmentation techniques to retrieve relevant segments by a textual query in a set of audio-visual recordings. Mettes *et al.* [23] proposed an approach using hierarchical clustering and syntactic and semantic similarity metrics to identify the segments.

While CodeTube also identifies fragments within a video, it is significantly different than those proposed in multimedia retrieval: it is not supervised and bases its segmentation algorithm on information specific to the software development domain, *i.e.,* the occurrence of code in the tutorials.

Use of Multimedia in Learning. Multimedia resources, especially those using videos, have been shown to be a very effective medium for learning, which is also often preferred by students over written text. Mayer [22] established twelve principles, based on numerous studies, that define the use and efficiency of multimedia in learning environments. Some of these principles clearly motivate CodeTube: (i) the *multimedia principle* states that people learn better from words and graphics than from words alone; (ii) the *temporal contiguity principle* indicates that people learn better when corresponding words and pictures are presented simultaneously rather than successively. Previous work has also shown that YouTube can be an efficient way to teach new concepts. Duffy [10] has shown that students like to use YouTube, as it provides a user-guided experience. For Mullamphy *et al.* [26] videos allow students to learn at their own pace. These observations were also confirmed within our first study.

7. CONCLUSION

We presented CodeTube, a novel approach to extract relevant fragments from software development video tutorials. CodeTube mixes several existing approaches and technologies like OCR and island parsing to analyze the complex unstructured contents of the video tutorials. Our approach extracts video fragments by merging the code information located and extracted within video frames, together with the speech information provided by audio transcripts. CodeTube also automatically complements the video fragments with relevant Stack Overflow discussions. We conducted two studies to evaluate CodeTube. Our survey highlighted the limitations of current video providers when it comes to dealing with software development. We received positive feedback about our approach and its potential. Also, we investigated the perception of our approach in industry environments by interviewing three leading developers, receiving useful insights on the strengths and potential extensions of our current work. To our knowledge, CodeTube is the first, and freely available[19] approach to perform video fragment analysis for software development.

The current approach can be ameliorated, and the fragments identification can be strengthened as well. We also plan to improve the user experience. Lastly, we plan to integrate additional sources of information other than Stack Overflow, towards the concept of a *holistic recommender*.

Acknowledgements. The authors would like to thank all the people involved in the evaluation of CodeTube. Ponzanelli is supported by the SNF Project "ESSENTIALS", No. 153129. Bavota is supported by the Free University of Bozen-Bolzano through the STREAM project (IN2036). Haiduc is supported in part by the NSF grant CCF-1526929.

[19] http://codetube.inf.usi.ch

8. REFERENCES

[1] M. Acharya, T. Xie, J. Pei, and J. Xu. Mining API patterns as partial orders from source code: from usage scenarios to specifications. In *Proceedings of ESEC/FSE 2007 (6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering)*, pages 25–34. ACM, 2007.

[2] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. Information retrieval models for recovering traceability links between code and documentation. In *Proceedings of ICSM (16th IEEE International Conference on Software Maintenance)*, pages 40–51. IEEE CS Press, 2000.

[3] A. Bacchelli, A. Cleve, M. Lanza, and A. Mocci. Extracting structured data from natural language documents with island parsing. In *Proceedings of ASE 2011 (26th IEEE/ACM International Conference On Automated Software Engineering)*, pages 476–479, 2011.

[4] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[5] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: A search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, pages 681–682. ACM, 2006.

[6] R. P. L. Buse and W. Weimer. Synthesizing API usage examples. In *Proceedings of ICSE 2012 (34th International Conference on Software Engineering)*, pages 782–792. IEEE, 2012.

[7] J. Cordeiro, B. Antunes, and P. Gomes. Context-based recommendation to support problem solving in software development. In *Proceedings of RSSE 2012 (3rd International Workshop on Recommendation Systems for Software Engineering)*, pages 85–89. IEEE Press, 2012.

[8] D. Cubranic, G. Murphy, J. Singer, and K. Booth. Hipikat: A project memory for software development. *IEEE TSE*, 31(6):446–465, 2005.

[9] T. Du, Y. Junsong, and D. Forsyth. Video event detection: From subvolume localization to spatiotemporal path search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(2):404–416, Feb. 2014.

[10] P. Duffy. Engaging the youtube google-eyed generation: Strategies for using web 2.0 in teaching and learning. In *European Conference on ELearning, ECEL*, pages 173–182, 2007.

[11] P. Galuščáková and P. Pecina. Experiments with segmentation strategies for passage retrieval in audio-visual documents. In *Proceedings of ICMR 2014 (4th International Conference on Multimedia Retrieval)*, pages 217:217–217:224. ACM, 2014.

[12] M. Goldman and R. C. Miller. Codetrail: Connecting source code and web resources. *Journal of Visual Languages & Computing*, 20(4):223–235, Aug. 2009.

[13] R. Holmes and A. Begel. Deep intellisense: A tool for rehydrating evaporated information. In *Proceedings of MSR 2008 (5th IEEE International Working Conference on Mining Software Repositories)*, pages 23–26, New York, NY, USA, 2008. ACM.

[14] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *Proceedings of ICSE 2005 (27th International Conference on Software Engineering*, pages 117–125. ACM, 2005.

[15] R. Holmes, R. J. Walker, and G. C. Murphy. Approximate structural context matching: An approach to recommend relevant examples. *IEEE Transactions on Software Engineering*, 32(12):952–970, Dec. 2006.

[16] I. Keivanloo, J. Rilling, and Y. Zou. Spotting working code examples. In *Proceedings of ICSE 2014 (36th International Conference on Software Engineering)*, pages 664–675. ACM, 2014.

[17] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of FSE 2006 (14th ACM SIGSOFT International Symposium on Foundations of Software Engineering)*, pages 1–11. ACM, 2006.

[18] O. Kononenko, D. Dietrich, R. Sharma, and R. Holmes. Automatically locating relevant programming help online. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 127–134, 2012.

[19] L. MacLeod, M.-A. Storey, and A. Bergen. Code, camera, action: How software developers document and share program knowledge using YouTube. In *Proceedings of ICPC 2015 (23rd IEEE International Conference on Program Comprehension)*, 2015.

[20] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman. Jungloid mining: Helping to navigate the api jungle. In *Proceedings of PLDI 2005 (16th ACM SIGPLAN Conference on Programming Language Design and Implementation)*, pages 48–61. ACM, 2005.

[21] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[22] R. E. Mayer. *Multimedia Learning*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[23] P. Mettes, J. C. van Gemert, S. Cappallo, T. Mensink, and C. G. Snoek. Bag-of-fragments: Selecting and encoding video fragments for event detection and recounting. In *Proceedings of ICMR 2015 (5th ACM on International Conference on Multimedia Retrieval)*, pages 427–434. ACM, 2015.

[24] L. Moonen. Generating robust parsers using island grammars. In *Proceedings of WCRE 2001 (8th Working Conference on Reverse Engineering)*, pages 13–22. IEEE CS, 2001.

[25] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, and A. Marcus. How can I use this method? In *Proceedings of ICSE 2015 (37th IEEE/ACM International Conference on Software Engineering)*, pages 880–890, 2015.

[26] D. Mullamphy, P. Higgins, S. Belward, and L. Ward. To screencast or not to screencast. *Anziam Journal*, 51:C446–C460, 2010.

[27] A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter Publishers, 1992.

[28] G. Petrosyan, M. P. Robillard, and R. De Mori. Discovering information explaining api types using text classification. In *Proceedings of ICSE 2015 (37th*

ACM/IEEE International Conference on Software Engineering), pages 869–879, 2015.

[29] L. Ponzanelli. Holistic recommender systems for software engineering. In *Proceedings of ICSE 2014 (36th ACM/IEEE International Conference on Software Engineering), Doctoral Symposium*, pages 686–689. ACM, 2014.

[30] L. Ponzanelli, A. Bacchelli, and M. Lanza. Leveraging crowd knowledge for software comprehension and development. In *Proceedings of CSMR 2013 (17th European Conference on Software Maintenance and Reengineering)*, CSMR '13, pages 57–66. IEEE Computer Society, 2013.

[31] L. Ponzanelli, A. Bacchelli, and M. Lanza. Seahawk: Stack overflow in the ide. In *Proceedings of ICSE 2013 (37th International Conference on Software Engineering)*, pages 1295–1298. IEEE Press, 2013.

[32] L. Ponzanelli, G. Bavota, M. di Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of MSR 2014 (11th Working Conference on Mining Software Repositories)*, pages 102–111. ACM Press, 2014.

[33] L. Ponzanelli, A. Mocci, and M. Lanza. Stormed: Stack overflow ready made data. In *Proceedings of MSR 2015 (12th Working Conference on Mining Software Repositories)*, pages 474–477. ACM Press, 2015.

[34] S. P. Reiss. Semantics-based code search. In *Proceedings of ICSE 2009 (31st International Conference on Software Engineering)*, pages 243–253. IEEE CS Press, 2009.

[35] P. C. Rigby and M. P. Robillard. Discovering essential code elements in informal documentation. In *Proceedings of ICSE 2013 (35th International Conference on Software Engineering)*, pages 832–841. IEEE Press, 2013.

[36] M. P. Robillard and Y. B. Chhetri. Recommending reference API documentation. *Empirical Software Engineering*, pages 1–29, 2014.

[37] N. Sawadsky and G. C. Murphy. Fishtail: From task context to source code examples. In *Proceedings of TOPI 2011 (1st Workshop on Developing Tools As Plug-ins)*, pages 48–51. ACM, 2011.

[38] J. Stylos and B. A. Myers. Mica: A web-search tool for finding api components and examples. In *Proceedings of the Visual Languages and Human-Centric Computing*, VLHCC '06, pages 195–202. IEEE Computer Society, 2006.

[39] S. Subramanian, L. Inozemtseva, and R. Holmes. Live api documentation. In *Proceedings of ICSE 2014 (36th International Conference on Software Engineering)*, pages 643–652. ACM, 2014.

[40] W. Takuya and H. Masuhara. A spontaneous code recommendation tool based on associative search. In *Proceedings of SUITE 2011 (3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation)*, pages 17–20. ACM, 2011.

[41] S. Thummalapenta and T. Xie. Parseweb: A programmer assistant for reusing open source code on the web. In *Proceedings of the ASE (22nd IEEE/ACM International Conference on Automated Software Engineering)*, pages 204–213, New York, NY, USA, 2007. ACM.

[42] M. Umarji, S. Sim, and C. Lopes. Archetypal internet-scale source code searching. In B. Russo, E. Damiani, S. Hissam, B. Lundell, and G. Succi, editors, *Open Source Development, Communities and Quality*, volume 275 of *IFIP The International Federation for Information Processing*, pages 257–263. Springer US, 2008.

[43] Z. Wen and V. Tzerpos. An effectiveness measure for software clustering algorithms. In *Proceedings of the 12th IEEE International Workshop on Program Comprehension*, pages 194–203, 2004.