UIScreens - Extracting User Interface Screens from Mobile Programming Video Tutorials

Mohammad Alahmadi[1,2], Ahmad Tayeb[1,3], Abdulkarim Khormi[1,4], Esteban Parra[1], Sonia Haiduc[1]
[1]Florida State University, Tallahassee, FL, United States
[2]University of Jeddah, Jeddah, Saudi Arabia
[3]King Abdulaziz University, Jeddah, Saudi Arabia
[4]Jazan University, Jizan, Saudi Arabia
{alahmadi, tayeb, khormi, parrarod, shaiduc}@cs.fsu.edu

ABSTRACT

Mobile apps are one of the most widely used types of software systems in existence today and more programmers and students learn how to develop them everyday. One of the most popular resources for learning mobile programming are videos hosted on social platforms such as YouTube. While useful, this type of resource has also its limitations, especially when developers are looking for user interface (UI) designs for mobile applications, since these are hard to search for and locate in videos.

We propose UIScreens, a web-based analysis and search engine that analyzes the visual contents of mobile programming video tutorials, then identifies and extracts the UI screens displayed in the videos. Our tool offers features such as searching for UI screens in videos, displaying an overview of the UI screens identified in a video under each search result, and navigating to the part of a video where a particular UI screen is being displayed and discussed. In a user study, participants agreed that UIScreens is usable and useful to quickly skim through videos, while the UI screens it extracts can help developers further determine the relevance of videos to a search topic.

CCS CONCEPTS

• **Software and its engineering** → *Documentation*; • **Computer vision** → *Image recognition*; • **Computer systems organization** → *Neural networks*;

KEYWORDS

Programming video tutorials, Android, iOS, Software documentation, Deep learning, Video mining

ACM Reference Format:
Mohammad Alahmadi[1,2], Ahmad Tayeb[1,3], Abdulkarim Khormi[1,4], Esteban Parra[1], Sonia Haiduc[1]. 2020. UIScreens - Extracting User Interface Screens from Mobile Programming Video Tutorials. In *Proceedings of The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

ESEC/FSE 2020, 8 - 13 November, 2020, Sacramento, California, United States

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Typically, the process of creating a mobile app is composed of (i) designing a set of user interface (UI) screens and (ii) implementing the functionality behind the UI. Designing an effective UI is one of the most difficult tasks in mobile development [3, 10, 11] and developers often rely on online resources to aid them in this task.

Recent studies found that videos are increasing in popularity as a resource for programming knowledge, given their efficient way to deliver information [6, 8, 9, 13]. YouTube is one of the most popular video-hosting platforms, as it hosts millions of videos on various topics, including mobile programming tutorials. Given the large amount of videos available, it is important to provide descriptive information about each video to help developers search and navigate through them. YouTube displays some metadata (*i.e.*, title and description, as provided by the video creators) about each video to make it easier for users to understand what a video is about. Unfortunately, this information is often insufficient for a developer to fully understand if a video contains specific information relevant to a programming task [12–14]. This is true in the case of developers looking for examples of mobile UI designs, as the only ways to determine if a video contains the right UI are to watch the videos entirely, leading to wasted time, or to skim through them, risking to miss information. While previous work has identified and addressed the problem of extracting and searching for mobile UI screens in apps hosted on mobile stores [3, 5], there is no prior work on assisting developers with this task in video tutorials. An advantage of searching for UI screens in videos is the fact that they can also help developers implement the relevant UI screens they identify by providing step-by-step instructions for their implementation [8].

In this paper we address the challenge of helping developers easily search and access UI screens in mobile programming video tutorials. More specifically, we introduce UIScreens, a tool that leverages Convolutional Neural Networks (CNN), object detection techniques, and an efficient search engine to detect, extract, and search for mobile UI screens in videos. In a user study with 16 participants, UIScreens was perceived as an easy to use tool that is useful for developers to search and explore UI designs in mobile programming video tutorials. Our tool is freely available to use online[1] and our replication package[2] contains the scripts, pre-trained models, and data behind our tool. A ***demo*** of our tool can be found here: **http://serenelab.cs.fsu.edu/uiscreens/demo**.

[1]http://serenelab.cs.fsu.edu/uiscreens
[2]https://zenodo.org/record/3901578

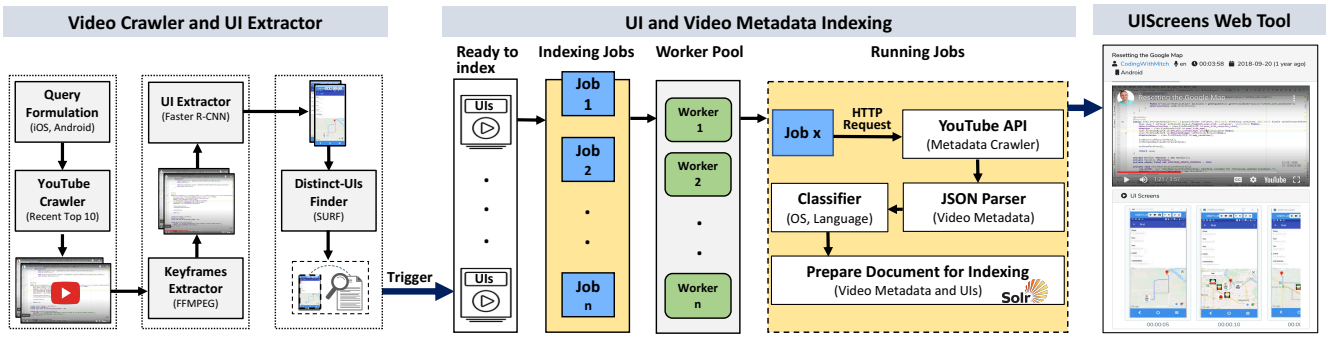

Figure 1: An overview of UIScreens and its main components

2 ARCHITECTURE

As Figure 1 depicts, UIScreens is composed of three main modules. The first two work together in the backend to automatically collect new videos and then extract and index UI layouts. This information is then used by and displayed in the UIScreens web-based tool depicted in the third module in Figure 1, which allows developers to preview and search for UI screens in video tutorials. In the following subsections we provide more details about each module.

2.1 Video Crawler and UI Extractor

The first module in our tool is responsible for crawling mobile programming video tutorials from YouTube and then identifying and extracting UI screens from them. In order to keep our video collection current and grow it with the latest top videos on various mobile programming topics without the need for human intervention, we automated the crawling process as follows. We first started by defining a set of 40 queries for YouTube, which focus on main topics of interest in Android and iOS programming. The first author formulated the queries based on his experience of more than 10 years with mobile programming, and the complete list of queries is available in our replication package. We then wrote a script that queried YouTube and crawled the top 50 video results for each of the 40 queries, resulting in approximately 2,000 unique videos (some of the videos may have appeared in the results of more than one query). After that, we designed the script such that the same YouTube queries are re-run every week, adding also a filter such that only videos that were added to YouTube in the last week are displayed in the results. We then collect the top returned videos for each query (up to 10 if available), download them at their highest quality and add them to our database. The process we followed ensures that our video collection contains the most relevant videos for each topic, while also growing with recent videos, rather than older, potentially obsolete ones. Currently, there are 2,915 videos hosted by our tool and this number is growing every week.

After a video is added to the UIScreens collection, a set of frames is extracted from it using FFMPEG[3], at the rate of one frame per second. UIScreens then locates and extracts the mobile UIs embedded in the frames of the video. To determine the precise location of each UI layout displayed in a frame (in the form of a rectangular *bounding box* that surrounds it), we adopt a deep learning object detection framework, namely Faster R-CNN [15], with Inception-Resnet V2 [16]. This extracts image features through the use of convolutional layers with different filters. To use the Faster R-CNN object detector, we trained it with data specific to our problem, namely UI bounding boxes within video frames. Due to space limitations, we refer the reader to our recent technical paper for details about the training of the model [1]. Our pre-trained model is also available in our replication package.

The list of frames we extracted from the videos in our collection are then fed into the pre-trained Faster R-CNN model, which outputs a set of bounding boxes for each UI screen that is identified in the frames. We then crop the UI screens based on their bounding box location. All cropped UIs are renamed with the *second* when they appeared in the video, determined based on the frame rate. This is important for a later stage in our tool, which enables users to navigate to the second where a UI appeared in a video (this is further explained in Section 2.3).

After all UIs are extracted for a video, another script gets triggered to remove duplicate UI screens. The script uses the Speeded Up Robust Features (SURF) algorithm [2] to extract and compare the feature vectors of pairs of UI screens using Euclidean distance. If two UIs are similar beyond a threshold (*i.e.*, 0.80), the first UI is removed. As extracting features from images is computationally intensive, we efficiently implemented the script to extract the features only once for each video and save them for further comparisons. We currently have 20,424 non-duplicate UI screens in our collection.

Lastly, Optical Character Recognition (OCR) is applied on the remaining UIs to extract the text appearing in them. We used Gdrive[4], which was recently found to work the best for programming screencasts [7]. We use the OCRed text as an additional source of information when indexing the videos in the next module of our tool, in order to allow developers to directly search for elements appearing in mobile UI screens (*e.g.*, *login*).

After the last step in this first module is completed, it triggers the second module in UIScreens, which indexes the videos based on their metadata and OCRed information. The whole UIScreens process involving the first two modules is repeated every week in order to accommodate the new videos being added to the database.

[3]https://ffmpeg.org/

[4]https://www.google.com/drive/

2.2 UI and Video Metadata Indexing

The input for the second module in UIScreens is a set of video IDs with their extracted UI layouts and OCRed text. This module is triggered by the first module, which typically passes a large number of videos that need to be indexed. Processing the entire video set would be slow and prone to failures since each video requires several steps and a third-party HTTP request, as shown in section *Running Jobs* of Figure 1. To avoid issues, we created a PHP script that processes subsets of videos through several scheduled indexing jobs. These indexing jobs are executed by workers that can work in parallel to perform specific tasks (Job x), as follows. First, a worker sends an HTTP request to the YouTube API to fetch the video's metadata using the video ID associated with the job. Note that YouTube limits the number of daily API calls that can be requested; therefore, if the daily limit is reached and YouTube rejects the request, our server releases the job and reschedules it to run after 24 hours, since YouTube will reset the API daily limit at that time. Second, video metadata is parsed through a JSON parser that extracts the video's title, description, published date, channel title, channel ID, audio language, tags, and duration. Third, we created a custom classifier that detects the mobile OS discussed in the video (Android, iOS, or cross-platform) from the title, description, and tags of the video. For this task, we manually created a list of common keywords describing each mobile platform (*e.g.*, Android keywords: Android, Java, etc.; iOS keywords: SWIFT, Xcode, etc.; cross-platform keywords: Xamrin, Flutter, etc.) and used string matching to count the number of matching terms corresponding to each platform in a video's title, description, and tags. The platform having the maximum match count is assumed to be the platform discussed in the video. The complete list of keywords we used for each platform is available in our replication package.

We also created another classifier for detecting the spoken language in the video. Our collection includes various videos in different languages and we want to offer developers a way to filter videos based on the language. While YouTube allows video creators to manually enter the audio language of the video, they often do not enter one, and English is assigned by default. Therefore, if the audio language of a video is not English, we assume the creator entered it and it is correct. If, however, the language is indicated as English, we want to ensure it was not incorrectly assigned by default, and we feed the title and description to an off-the-shelf language detection package[5] that can detect up to 112 languages.

We then use Apache Solr[6] to index the metadata information, OCRed UI text, and the extracted UI screens of each video. We used DisMax as the query parser and assigned the following weights to the metadata: title - 3, description - 2, tags - 1, and OCRed text - 1.

2.3 UIScreens Web Tool

UIScreens[7] has a web frontend implemented using PHP with the Laravel framework. We explain the main features of UIScreens shown in Figure 2 as follows.

A. Dynamic Search: Users can formulate their query and as they type in the search bar, our search engine suggests a list of topics

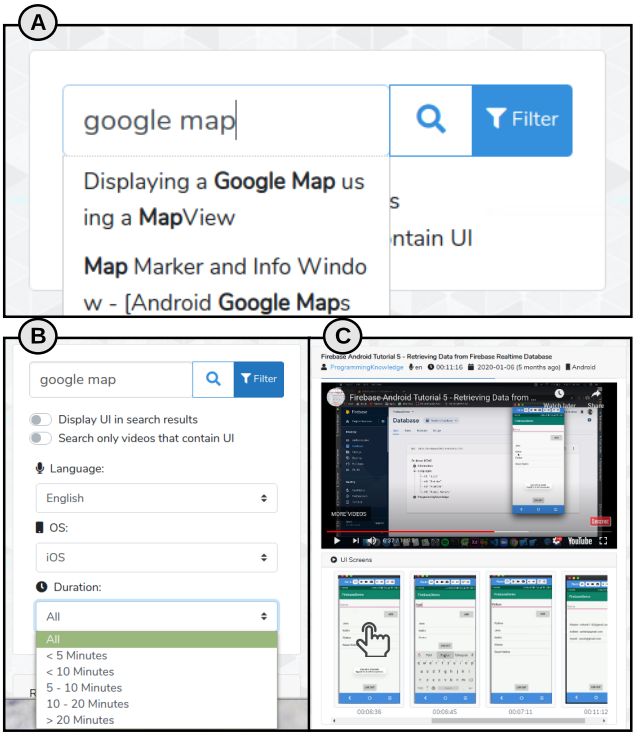

Figure 2: UIScreens: web frontend

based on the titles of videos stored in our database.

B. Search Filters: We defined five search filters as follows: (i) *"Display UI in search results"*, which allows users to see the extracted UIs below each video in the search results, (ii) *"Search only videos that contain UI"*, so users can limit their search to only videos that contain a UI design, (iii) *"Language"*, where users can select one of the languages detected by our classifier, (iv) *"OS"*, so users can focus their search on their choice of three mobile OS platforms (iOS, Android, and cross-platform), and (v) *"Duration:"*, where users can limit their search to videos of a certain duration.

C. Video Details: This page appears after a user clicks on a video of interest. The title of the video is shown on the top along with other metadata such as the video author. The video is then displayed, along with the list of extracted UI screens Users can click on a specific UI screen to navigate directly to the part of the video where the UI appears in the video.

3 EVALUATION

In this section, we describe the design and results of a user study with developers evaluating UIScreens based on its *usefulness*, *accuracy*, and *usability*.

Study Design: The study was conducted via an anonymous online survey (available in our replication package[8]). having five sections: (i) ***Background***, (ii) ***Usefulness***, (iii) ***Accuracy***, (iv) ***Usability***, and (v) ***Overall Feedback***. The *first section* of the survey collects background demographic information about the occupation of the participants and their mobile development experience. Participants had to have Android or iOS programming experience to

[5]https://github.com/patrickschur/language-detection
[6]https://lucene.apache.org/solr/
[7]http://serenelab.cs.fsu.edu/uiscreens/
[8]https://zenodo.org/record/3901578

be qualified to take the survey and move to the next section. In the remaining sections, the participants could express their assessment of particular aspects of UIScreens based on a 5 point Likert scale. *Section 2* of the study first focused on getting participants familiar with UIScreens, by having them run several queries, observing the results, and trying out the different features it offers. It then asked participants to evaluate the usefulness of each of the features in UIScreens, *i.e.*, displaying the UI screens alongside videos in the search results and detailed view of a video, filtering results based on the spoken language, the presence of UI screens, the OS, or the duration of a video, and being able to navigate to where a UI screen appears in a video by clicking on it. The section also asked participants if they believed that extracting and displaying the UI screens below a video can help them decide if it is relevant to a search. *Section 3* of the survey evaluated the accuracy of the UI identification, extraction, and duplicate removal. It first provided participants with a randomly selected short video to watch (<5 min) containing UI screens and asked them if the extracted screens were sufficient to understand the main UI design elements discussed in the video, and if the list of UI screens contained duplicates. It then provided participants with another short video, this time randomly selected from the set of videos for which our tool did not identify any UI screens to extract. The participants were asked to watch the video entirely and then indicate if they found any UI screens that our tool failed to identify. *Section 4* evaluates the usability of UIScreens by asking participants their agreement with six statements (three positive and three negative) based on the SUS usability scale by Brooke [4]. The positive statements were: *I would like to use UIScreens to identify potential UI screens useful to my tasks*", "*I found the various features in UIScreens well-integrated*", and "*I imagine that most people would learn how to use UIScreens quickly*". The negative statements were: "*UIScreens is unnecessarily complex*", "*UIScreens is very cumbersome to use*", and "*I needed to learn a lot of new things to use UIScreens*". These statements are similar to those used to evaluate usability in previous work [3]. *Section 5* collects feedback from the participants, asking them about their overall impression of UIScreens and any features they would like to see integrated.

Results: We received 16 valid answers to our survey (we eliminated answers where participants only spent a few minutes on the survey, which otherwise took at least 20 min to answer). We present the results of our user study below, divided by section. The complete results are available in our replication package.

- ***Background*:** All participants in our study had experience with Android development, and eight participants also had experience with iOS programming. The level of experience of our participants with mobile programming ranged between a few months and a few years. The participants included nine undergraduate students, one Masters' student, three Ph.D. students, one professional developer, one field operator, and one faculty member.
- ***Usefulness*:** In general, participants agreed that the vast majority of the features in UIScreens were useful. On a Likert scale of 1 to 5, where 5 indicates strong agreement that a feature is useful and 1 indicates strong disagreement, the main functionalities of UIScreens received the following scores on average (in descending order): navigating to where a UI screen is displayed in a video (4.63), filtering search results based on the presence of UI screens (4.44), displaying UIs below a video in the list of search results (4.38), filtering results based on the OS (4.38), displaying UIs below a video in its detailed view (4.31), filtering search results based on the spoken language (4.06), and filtering search results based on the video duration (3.5). Participants also indicated a very strong agreement (4.88 score on average) with the statement that extracting and displaying the UI screens below a video can help them decide if a video is relevant to a search.
- ***Accuracy*:** When asked about the sufficiency of the extracted UI screens to understand the main UI design elements discussed in the video, the participants assigned on average a score of 3.31, which is slightly positive, between "Neither agree nor disagree" (3) and "Agree" (4). However, when analyzing the results in more detail, we noticed that half of the participants either agreed or strongly agreed that the extracted UI screens were sufficient, 25% of the participants had a neutral opinion, and only 25% found the extracted UI screens insufficient. In regards to a statement about no duplicate UI screens being extracted, the average score was also 3.31 on average, indicating that the participants slightly agreed that the list of UI screens does *not* contain duplicate information. Half the participants considered that the extracted UI screens did not present duplicate information (*i.e.*, all UI screens presented are distinct), whereas 31% of them disagreed and the remaining 19% had a neutral view. 14 of the 16 participants further answered that they did not find any UIs in the video for which UIScreens had not detected any UI screens. The remaining two participants indicated "*not sure*" about the video containing UIs. These results indicate that, while UIScreens did not miss any UIs in a video, there is a room for improving the UI duplicate removal to select only sufficient and distinct UIs. We also believe that extracting more specific UI design components from the screens in the future could help developers better understand the UI design elements discussed in the video.
- ***Usability*:** Participants generally agreed with the positive statements about UIScreens' usability (scores of 4.38, 4.19, and 4.44 on average), while at the same time they generally disagreed with the negative statements (scores of 2.13, 1.88, and 1.75 on average). The results indicate that UIScreens is intuitive and easy to use.
- ***Overall Feedback*:** The score for the overall participants' impression about UIScreens was 4.44 out of 5 on average, indicating an overwhelmingly positive opinion. Some of features the participants would like to see integrated include automatically recognizing different types of UI components in the screens, ability to expand the UI screens for a more detailed view, and being able to toggle on and off the view of the UI screens for particular videos. We plan to address these suggestions in our future work.

4 CONCLUSION

We introduced UIScreens, a web-based tool that extracts, indexes, displays, and searches mobile UI layouts embedded in video programming tutorials. UIScreens provides an intuitive user experience and several search and navigation features that were found useful by participants in a user study.

5 ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation grants CCF-1846142 and CCF-1644285.

REFERENCES

[1] Mohammad Alahmadi, Abdulkarim Khormi, and Sonia Haiduc. 2020. UI screens identification and extraction from mobile programming screencasts. In *Proceedings of the 28th Conference on Program Comprehension*. ACM, 222–232.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. 2006. SURF: Speeded up robust features. In *European conference on computer vision*. Springer, 404–417.

[3] Carlos Bernal-Cardenas, Kevin Moran, Michele Tufano, Zichang Liu, Linyong Nan, Zhehan Shi, and Denys Poshyvanyk. 2019. Guigle: A GUI search engine for Android apps. *arXiv preprint arXiv:1901.00891* (2019).

[4] John Brooke. 1996. SUS - A quick and dirty usability scale. *Usability Evaluation In Industry* (1996), 7.

[5] Chunyang Chen, Sidong Feng, Zhenchang Xing, Linda Liu, Shengdong Zhao, and Jinshui Wang. 2019. Gallery DC: Design Search and Knowledge Discovery through Auto-created GUI Component Gallery. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–22.

[6] Javier Escobar-Avila, Deborah Venuti, Massimiliano Di Penta, and Sonia Haiduc. 2019. A survey on online learning preferences for computer science and programming. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 170–181.

[7] Abdulkarim Khormi, Mohammad Alahmadi, and Sonia Haiduc. 2020. A Study on the Accuracy of OCR Engines for Source Code Transcription from Programming Screencasts. In *Proceedings of the 17th IEEE/ACM Working Conference on Mining Software Repositories*. 376–386.

[8] Laura MacLeod, Andreas Bergen, and Margaret-Anne Storey. 2017. Documenting and sharing software knowledge using screencasts. *Empirical Software Engineering* 22, 3 (June 2017), 1478–1507. https://doi.org/10.1007/s10664-017-9501-9

[9] Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. 2015. Code, camera, action: How software developers document and share program knowledge using YouTube. In *Proceedings of the 23rd IEEE International Conference on Program Comprehension (ICPC'15)*. Florence, Italy, 104–114.

[10] Kevin Moran, Boyang Li, Carlos Bernal-Cárdenas, Dan Jelf, and Denys Poshyvanyk. 2018. Automated reporting of GUI design violations for mobile apps. *arXiv preprint arXiv:1802.04732* (2018).

[11] Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse engineering mobile application user interfaces with remaui (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 248–259.

[12] Esteban Parra, Javier Escobar-Avila, and Sonia Haiduc. 2018. Automatic tag recommendation for software development video tutorials. In *Proceedings of the 26th Conference on Program Comprehension*. ACM, 222–232.

[13] Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Massimiliano Di Penta, Rocco Oliveto, Barbara Russo, Sonia Haiduc, and Michele Lanza. 2016. CodeTube: Extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th ACM/IEEE International Conference on Software Engineering (ICSE'16)*. ACM, Austin, TX, 645–648.

[14] Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Rocco Oliveto, Massimiliano Di Penta, Sonia Cristina Haiduc, Barbara Russo, and Michele Lanza. 2019. Automatic identification and classification of software development video tutorial fragments. *IEEE Transactions on Software Engineering* (2019). https://doi.org/10.1109/TSE.2017.2779479

[15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. *arXiv:1506.01497 [cs]* (June 2015). http://arxiv.org/abs/1506.01497 arXiv: 1506.01497.

[16] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. 2016. Inception-v4, Inception-ResNet and the impact of residual connections on learning. *arXiv:1602.07261 [cs]* (Feb. 2016). http://arxiv.org/abs/1602.07261 arXiv: 1602.07261.