scvRipper: Video Scraping Tool for Modeling Developers' Behavior Using Interaction Data

Lingfeng Bao[1], Jing Li[2], Zhenchang Xing[2], Xinyu Wang [§1], and Bo Zhou[1]

[1]*College of Computer Science, Zhejiang University, Hangzhou, China*
[2]*School of Computer Engineering, Nanyang Technological University, Singapore*
{lingfengbao, wangxinyu, bzhou}@zju.edu.cn; {jli030, zcxing}@ntu.edu.sg;

***Abstract*—Screen-capture tool can record a user's interaction with software and application content as a stream of screenshots which is usually stored in certain video format. Researchers have used screen-captured videos to study the programming activities that the developers carry out. In these studies, screen-captured videos had to be manually transcribed to extract software usage and application content data for the study purpose. This paper presents a computer-vision based video scraping tool (called *scvRipper*) that can automatically transcribe a screen-captured video into time-series interaction data according to the analyst's need. This tool can address the increasing need for automatic behavioral data collection methods in the studies of human aspects of software engineering.**

Demo video: https://www.youtube.com/watch?v=DElYOhids8Y

I. Introduction

An important area in studying human aspects of software engineering is to model and analyze the developers' information needs and behavior patterns in different software development tasks [1, 2].This is important for informing the design of new kind of programming tools tailored to observed patterns and strategies [3]

Researchers have used human observer, think aloud, software instrumentation, and screen-capture videos to collect developers' behavior data in software development tasks. Among these data collection methods, screen-captured videos provide an unobtrusive, generic and easy-to-deploy method to record the developers' interaction with not only the IDE and the program but also with other software tools and application content (e.g., web browsers to search online resources, search queries issued, web page visited) that the developers use in software development.

To perform quantitative analysis of developers' behavior, researchers had to manually transcribe screen-captured videos into software usage and application content data. The ratio of video recording time and analysis time was reported to be 1:4 - 1:7. For example, Ko and Myers [3] reported "... analysis of video data by repeated rewinding and fast-forwarding..." in their study of software errors in programming systems.

As the amount of research on human aspects of software engineering has increased in recent years, there has been a greater need to come up with a solution to automatically extract software usage and application content data from screen-captured videos. This technique will facilitate the modeling and analysis of developers' behavior in software engineering research and practices.

This paper presents our computer-vision-based video scraping technique and tool (called *scvRipper*). The *scvRipper* tool can recognize window-based applications in screen-capture videos, and extract application content from the recognized application windows. We discuss the design and implementation of our *scvRipper* tool in Section II and Section III. We briefly describe a case study of the *scvRipper* tool in Section IV.

II. Tool Design

Fig. 1 presents the architecture of our video scraping tool. The *scvRipper* tool takes as input a screen-captured video, i.e., a stream of screenshots taken by screen-capture tools. It recognizes application windows in the screenshots based on the definition of application windows provided by the analyst. It produces as output time-series interaction data (i.e., software usage and application content over time) extracted from the screen-captured video.

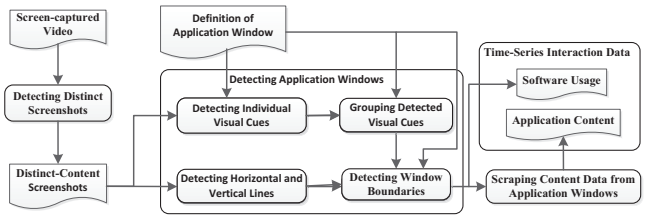


Fig. 1. The Architecture of Our Video Scraping Tool

A. Definition of Application Window

The definition of an application window "informs" the *scvRipper* tool with the window layout, the sample images of distinct visual cues of the window's GUI components, and the GUI components to be scraped once they are recognized. Fig. 2 shows the metamodel of application windows. The *scvRipper* tool assumes that an application window is composed of a hierarchy of GUIComponents. Rows and windows define the layout of the application window. A row or window can contain nested rows, nested windows, and/or leaf GUIItems. Rows and GUIItems have relative positions in the application window (denoted as $index$), while windows do not have. A GUIItem contains an order set of VisualCues. A

[§]Xinyu Wang is the corresponding author

978-1-4799-1934-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICSE.2015.220

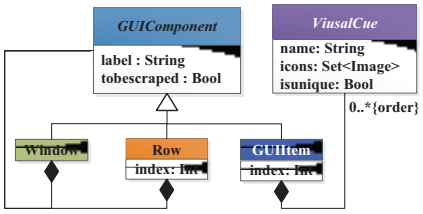


Fig. 2. The Metamodel of Application Windows

VisualCue contains a set of sample images of the visual cue. A VisualCue can be unique ($isunique = ture$) in an application window. The GUIComponents whose $tobescraped = true$ will be scraped from the application window.

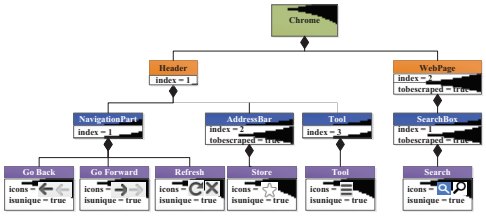
Fig. 3 shows partially the definition of the Eclipse IDE window and the Google Chrome Window. This definition of the Eclipse window assumes that the Eclipse window consists of a GUIItem (TitleBar) and four rows (Menu, ToolBar, MainContent, and StatusBar) from top down. We omit the definition details of Menu, ToolBar and StatusBar due to space limitation. The TitleBar contains a unique VisualCue (Eclipse application icon). MainContent row may contain CodeEditor windows and ConsoleView windows. CodeEditor window contains FileTab and EditArea GUIItems. These two GUIItems contain non-unique visual cues (such as Java file icons, compile error icons). This definition instructs the *scvRipper* tool to scrape CodeEditor and ConcoleView content from the Eclipse window.

The definition of the Chrome window assumes that the Chrome window consists of two rows from top down: Header and WebPage. The Header contains three GUIItems from left to right: NavigationPart, AddressBar, and Tool. NavigationPart contains three VisualCues from left to right: GoBack, GoForward, and Refresh buttons. These buttons are unique in the Chrome window. The WegPage may contain a SearchBox GUIItem as commonly seen in search engine webpages. A SearchBox has a unique Search button VisualCue. This definition instructs *scvRipper* to scrape AddressBar, SearchBox and WebPage from the Chrome window.

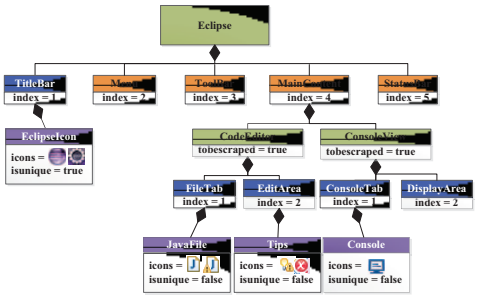
B. Video Scraping Process

The *scvRipper* tool essentially uses computer-vision techniques to transcribe a stream of screenshots that only human can interpret into a stream of interaction data that computer can automatically analyze or mine for behavioral patterns (see our tenical report [4]). First, the *scvRipper* tool uses image differencing technique [5] to detect screenshots with distinct content in the screen-captured video. This step filters out the screenshots with no differences due to no human-computer interaction or with only small differences due to mouse movement, button click or small scrolling.

Then, the core algorithm of the *scvRipper* tool processes one distinct-content screenshots at a time to recognize application windows in the screenshot in four steps: 1) detect horizontal and vertical lines, 2) detect individual visual cues, 3) group detected visual cues, and 4) detect window boundaries. The *scvRipper*'s algorithm can accurately recognize stacked or



(a) Definition of Google Chrome Window



(b) Definition of Eclipse IDE Window

Fig. 3. Two Instances of Application-Window Metamodel

side-by-side windows. The recognized application windows identify software used at a specific time in the video.

Finally, *scvRipper* scrapes the GUIComponent images from the recognized application windows in the screenshot as specified in the definition of application windows. It uses Optical-Char-Recognition (OCR) technique to convert the scraped GUIComponent images into textual application content that the developer used at a specific time in the video.

The upper part of Fig. 4 shows an illustrative example of a screen-captured video. Assume that a developer views two web pages side-by-side in the two Chrome windows. He then maximizes one of the Chrome windows. After a while, he switches from the Chrome window to an Eclipse IDE window. He opens two different files in Eclipse and read the code. Next he switches from the Eclipse window back to the Chrome window. In this example, four distinct-content screenshots can be identified at five time periods. Note that the screenshots at time periods $t_2 - t_3$ and $t_5 - t_6$ are the same.

The lower part of Fig. 4 shows the time-series interaction data that the *scvRipper* tool extracts from these four distinct-content screenshots. Bulky contents (e.g., web page, code fragment) are omitted due to space limitation. This time-series interaction data identifies the software tools that the developer used at different time periods. It also identifies the application content that the developer processed (such as search queries, websites visited, code fragments, and runtime exceptions) at differen time periods.

III. Tool Implementation

We have developed a configuration tool to aid the definition of application windows. The tool can define the hierarchy of GUIComponents, configure the attributes of GUIComponents, and attach sample images of visual cues to GUIComponents. Fig. 5 shows the screenshot of the configuration tool in

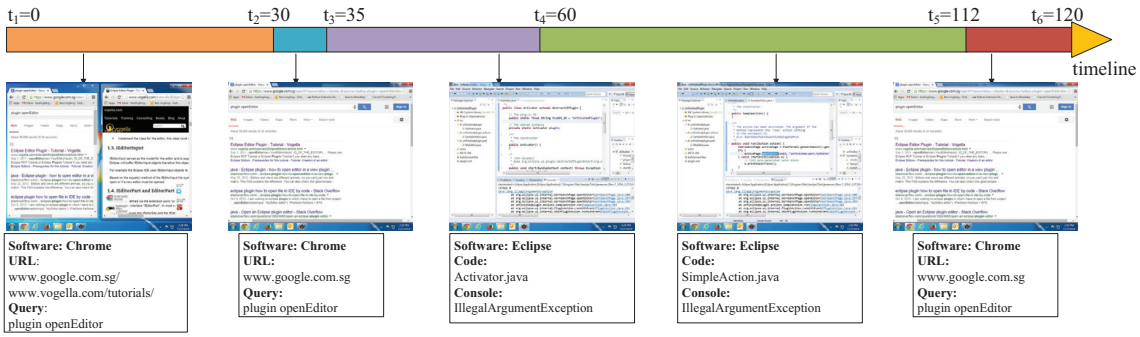


Fig. 4. An Illustrative Example Of a Screen-Captured Video and Video Scraping Results

defining the Eclipse IDE window and the Google Chrome window shown in Fig. 3.

Collecting sample images of visual cues often require certain efforts. However, this task usually need to be done only once. The definition of an application window can be applied to screen-captured videos taken in different screen resolutions and window color schema, as neither window definition nor computer-vision techniques that *scvRipper* uses are sensitive to screen resolutions and window color schema.

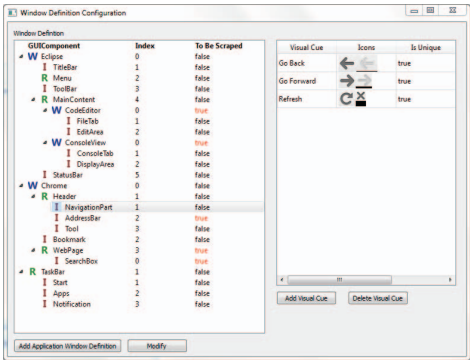


Fig. 5. The Configuration Tool for Window Definition

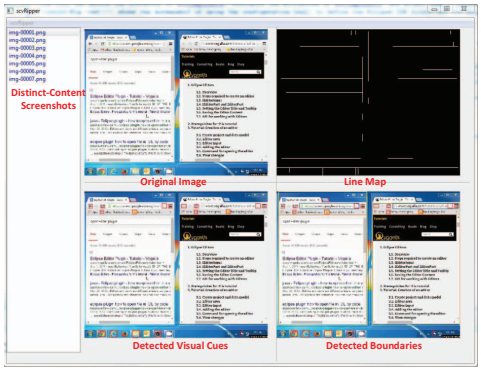


Fig. 6. The Screenshot of scvRipper

We have implemented our *scvRipper* tool using *OpenCV* (*opencv.org*, an open-source computer vision library). The core algorithm of the *scvRipper*) tool has been implemented using the *OpenCV*'s efficient computer-vision algorithms, such as candy edge map for detecting horizontal and vertical lines, keypoint based template matching for detecting individual visual cues, and normalized min-mix cut algorithm for grouping detected visual cues.

Fig. 6 shows the Graphical User Interface (GUI) of the *scvRipper* tool. The analyst can select a screen-captured video. The *scvRipper* tool parses the video and detect distinct-content screenshots. It lists the distinct-content screenshots in the left panel. The analyst can analyze one screenshot at a time or analyze all the screenshot in batch mode. The *scvRipper* tool visualizes the intermediate image processing results in the right panel, such as the detected horizontal and vertical lines, the detected visual cues, and the detected window boundaries. The analyst can zoom-in and inspect these intermediate results to determine the quality of the video-scraped data.

IV. Case Study

We evaluated the usefulness, effectiveness and runtime performance of our *scvRipper* tool using the 29 hours screen-captured task videos from our previous study [6]. Our previous study was to study developers' online search behavior during software development. It included two development tasks: 1) develop a new P2P chat software, and 2) fix bugs and extends an existing Eclipse editor plugin. 11 graduate students were recruited in the first task, and 13 different graduate students were recruited in the second task from the School of Computer Science, Fudan University. We briefly summarized our empirical results here. Interested readers are referred to our technical report [4] for detailed discussions .

Based on the time-series interaction data that the *scvRipper* tool extracted from the task videos, we performed two quantitative analysis of the participants' online search behavior during the two software development tasks. First, we computed a probabilistic model of the participants' search frequencies and intervals. Second, we studied the dynamics of the participants' working context over time. These two quantitative analysis demonstrated the usefulness of the video-scraped interaction data for modeling and analyzing developers' behavior.

We randomly sampled 500 distinct-content screenshots from different developers' task videos at different time periods. We qualitatively examined the intermediate image processing results of our *scvRipper* tool, including the representativeness of these distinct-content screenshots, the detected application windows in these sampled screenshots, and the OCR accuracy of the scraped search query keywords. Our analysis confirmed the effectiveness and accuracy of the *scvRipper* tool.

Our evaluation ran the *scvRipper* tool on a Window 7

computer with 4GB RAM and Intel(R) Core(TM)2 Duo CPU. Our results identified the bottleneck of the tool's runtime performance. The most time-consuming step of the *scvRipper* tool was the step for detecting individual visual cues. This step consumes about 97% of the processing time of distinct-content screenshots. The current tool implementation sequentially detects visual cues in a screenshot one at a time. The runtime performance of the *scvRipper* tool could be significantly improved by parallel computing [7] and hardware-implementation of template-matching algorithm [8].

V. Related Work

Computer vision techniques have been used to identify user interface elements from screen-captured images or videos. Prefab [9] models widgets layout and appearance of an user interface toolkit as a library of prototypes. A prototype consists of a set of parts (e.g., a patch of pixels) and a set of constraints regarding those parts. Prefab identifies the occurrence of widgets from a given prototype library in an image of an user interface by first assigning image pixels into parts from the prototype library and then filtering widget occurrences according to the part constraints. Waken [10] uses image differencing technique to identify the occurrence of cursors, icons, menus, and tooltips that an application contains in screen-captured videos. Sikuli [11] uses template matching techniques [12] to find GUI patterns on the screen.

These computer-vision based techniques inspired the design and implementation of our video scraping technique, including the metamodel of application windows, the detection of distinct-content screenshots, and the detection of application windows. These existing techniques have focused on visual search, GUI automation, and implementing new interaction techniques. In contrast, the *scvRipper* tool focuses on extracting time-series interaction data from screen-captured videos. Unlike the video data that only human can interpret, the extracted time-series interaction data can be automatically analyzed to discover behavioral patterns.

Instrumentation techniques [13, 14] can directly log a user's interaction with software tools and application content. They usually requires the support of sophisticated reflection APIs (e.g., Accessability API or UI Automation API) provided by applications, operating systems and GUI toolkits. Furthermore, a user can use several software tools (e.g., Eclipse IDE, different web browsers) in his work. Instrumenting all these software tools require significant efforts. The *scvRipper* tool provides a generic and easy-to-deploy solution to collect software usage and application content data across several applications.

Some work proposes to combine low-level operating system APIs and computer vision techniques to track human computer interaction. Hurst et al. [15] leverages image differencing and template matching techniques to improve the accuracy of target identification that the users click. Chang et al. [16] proposed a hybrid framework for detecting text blobs in user interface by combining pixel-based analysis and accessibility metadata of the user interface. In contrast, The *scvRipper* tool analyzes screen-captured videos without using accessibility information.

VI. Conclusion

This paper presented our computer-vision-based video-scraping tool (called *scvRipper*) that can automatically extract time-series interaction data from screen-captured videos. Our *scvRipper* tool is generic and easy to deploy. It can collect software usage and application content data across several applications according to the analyst's need. The extracted time-series interaction data can be used to quantitatively model and analyze developers' behavior during software development.

Our *scvRipper* tool can address the increasing need for automatic behavioral data collection methods in the studies of human aspects of software engineering. In the future we will improve the *scvRipper* tool's runtime performance using parallel computing and hardware acceleration. We are also interested in combining operating system level instrumentation (e.g., mouse and keystroke) with the core algorithm of *scvRipper* to collect more accurate time-series interaction data.

Acknowledgment

This research was supported by the National Basic Research Program of China (the 973 Program) under grant 2015CB352201, and National Key Technology R&D Program of the Ministry of Science and Technology of China under grant 2014BAH24F02. This work is supported by NTU SUG M4081029.020 and MOE AcRF Tier1 M4011165.020.

References

[1] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, 2006.

[2] J. Lawrance, C. Bogart, M. Burnett, R. Bellamy, K. Rector, and S. D. Fleming, "How programmers debug, revisited: An information foraging theory perspective," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 197–215, 2013.

[3] A. J. Ko and B. A. Myers, "A framework and methodology for studying the causes of software errors in programming systems," *J VISUAL LANG COMPUT*, vol. 16, no. 1, pp. 41–84, 2005.

[4] https://sites.google.com/site/jinglisites/home/project.

[5] D.-C. Wu and W.-H. Tsai, "Spatial-domain image hiding using image differencing," *Proc. ICCVISP*, vol. 147, no. 1, pp. 29–37, 2000.

[6] H. Li, Z. Xing, X. Peng, and W. Zhao, "What help do developers seek, when and how?," in *Proc. WCRE*, pp. 142–151, 2013.

[7] Q. Zhang, Y. Chen, Y. Zhang, and Y. Xu, "Sift implementation and optimization for multi-core systems," in *Proc. IPDPS*, pp. 1–8, 2008.

[8] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc, "Gpu-based video feature tracking and matching," in *EDGE, Workshop on Edge Computing Using New Commodity Architectures*, vol. 278, p. 4321, 2006.

[9] M. Dixon and J. Fogarty, "Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure," in *Proc. CHI*, pp. 1525–1534, 2010.

[10] N. Banovic, T. Grossman, J. Matejka, and G. Fitzmaurice, "Waken: reverse engineering usage information and interface structure from software videos," in *Proc. UIST*, pp. 83–92, 2012.

[11] T. Yeh, T.-H. Chang, and R. C. Miller, "Sikuli: using gui screenshots for search and automation," in *Proc. UIST*, pp. 183–192, 2009.

[12] D. A. Forsyth and J. Ponce, *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[13] D. M. Hilbert and D. F. Redmiles, "Extracting usability information from user interface events," *ACM Comput. Surv.*, vol. 32, no. 4, pp. 384–421, 2000.

[14] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. Wixon, "Tracking real-time user experience (true): a comprehensive instrumentation solution for complex systems," in *Proc. CHI*, pp. 443–452, 2008.

[15] A. Hurst, S. E. Hudson, and J. Mankoff, "Automatically identifying targets users interact with during real world tasks," in *Proc. IUI*, pp. 11–20, 2010.

[16] T.-H. Chang, T. Yeh, and R. Miller, "Associating the visual representation of user interfaces with their internal structures and metadata," in *Proc. UIST*, pp. 245–256, 2011.