VT-Revolution: Interactive Programming Tutorials Made Possible

Lingfeng Bao
Zhejiang University
China

Zhenchang Xing
Australian National University
Australia

Xin Xia
Monash University
Australia

David Lo
Singapore Management University
Singapore

Shanping Li
Zhejiang University
China

ABSTRACT

Programming video tutorials showcase programming tasks and associated workflows. Although video tutorials are easy to create, it is often difficult to explore the captured workflows and interact with the programs in the videos. In this work, we propose a tool named VTRevolution – an interactive programming video tutorial authoring system. VTRevolution has two components: 1) a tutorial authoring system leverages operating system level instrumentation to log workflow history while tutorial authors are creating programming video tutorials; 2) a tutorial watching system enhances the learning experience of video tutorials by providing operation history and timeline-based browsing interactions. Our tutorial authoring system does not require any special recording tools or instrumentation of target applications. Neither does it incur any additional burden on tutorial authors to add interactions to video tutorials. Given a video tutorial enriched with synchronously-logged workflow history, our tutorial watching system allows tutorial watchers to explore the captured workflows and interact with files and code in a way that is impossible for video data alone. We conduct a user study of 90 developers to evaluate the design and effectiveness of our system in helping developers learn programming knowledge in video tutorials.
Demonstration video link: https://youtu.be/9HSVQRaqgA0
Tool website: http://baolingfeng.xyz:8080/VTRevolution/

CCS CONCEPTS

• **Human-centered computing** → *User interface toolkits*; • **Software and its engineering** → *Software libraries and repositories*;

KEYWORDS

Workflow, Video Tutorial

ACM Reference Format:
Lingfeng Bao, Zhenchang Xing, Xin Xia, David Lo, and Shanping Li. 2018. VT-Revolution: Interactive Programming Tutorials Made Possible. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3236024.3264587

1 INTRODUCTION

Programming video tutorials have become a very popular resource to help developers learn coding. A survey reported that more than 2/3 developers use video tutorials weekly and monthly [17]. Video tutorials allow developers to directly observe the workflows that are carried out to complete programming tasks, which might be more valuable than traditional, text-based learning resources [12].

Comparing with text-based tutorials, it's much easier to record a 5 minutes tutorial video using a screen-capturing tool like Snagit[1] [16]. However, due to the nature of video content (i.e., a stream of screen-captured images), it is difficult to let video tutorial watchers interact with the tutorial content. First, a tutorial watcher cannot get an holistic, high-level overview of workflow, e.g., when to modify which file and how, or what content the tutorial author already adds to a file up till now. Second, there is no effective search and navigation support for tutorial content. Finally, it is inconvenient to access complementary learning resources. A tutorial watcher is usually unfamiliar with some APIs in the tutorial and need complementary learning resources (e.g., API documentation) to assist their learning of the tutorial.

The above interaction limitations can lead to misunderstandings of the content [10], difficulties in keeping up with the pace of the tutorial and reduced knowledge retention [15]. Existing multimedia tutorial authoring tools (e.g. *HyperCard*[2], *Adobe Authorware*[3]) allow the authors to create more interactive video tutorials and annotate the video content with rich information. But the limitation is that they require authors define the story line and there are no aids for freely exploring the captured workflows. Some recent work [1, 2, 17] uses Optical-Char-Recognition (OCR) techniques to convert video content into text, which can then be summarized, searched or linked to other resources. But OCR technique has limitations in time cost and quality of extracted data.

In this paper, we design and implement an interactive programming video tutorial system named *VT-Revolution*. A key design goal of *VT-Revolution* is, on the one hand, we want tutorial watchers to be able to freely explore the captured workflow in the tutorial and interact with the tutorial content; and, on the other hand, we do not want to create a significant burden on tutorial authors to

[1]https://www.techsmith.com/snagit.html
[2]http://hypercard.org/
[3]http://www.adobe.com/products/authorware/

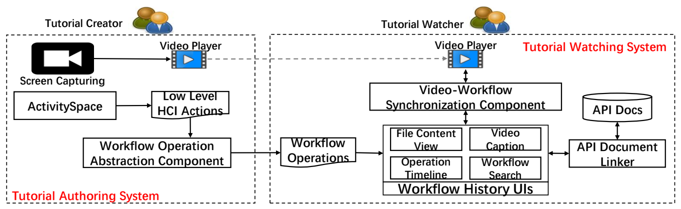


Figure 1: The Framework of *VT-Revolution*

manually annotate tutorial videos with workflow history and other complementary resources.

To achieve this goal, our tutorial authoring system records screen-captured video and uses ActivitySpace [3, 6] to synchronously log the workflow history while the author is interacting with software development environment to create a programming tutorial. The ActivitySpace tool can unobtrusively track and analyze a developer's interactions with a wide range of software tools and applications commonly used in software development, including IDEs, text editors, web browsers, and office software. It has been successfully applied in several studies, e.g., [5, 20]. As the first proof-of-concept of our interactive programming video tutorial system, we only log tutorial authors' actions within the IDE. This is because the programming video tutorial author usually spends most of the time in the IDE to demonstrate how to complete the programming task. ActivitySpace allows us to easily extend the workflow history logging to other software tools used during the programming tutorial.

Figure 1 presents our *VT-Revolution* system, which contains a tutorial authoring system and a tutorial watching system. The tutorial authoring system integrates a regular screen-capturing tool (e.g., Snagit), the ActivitySpace framework [3, 6], and a workflow operation abstraction component. The tutorial watching system takes as input an interactive programming tutorial, consisting of a screen-captured programming video and a synchronously-logged time-series of workflow operations. The tutorial watching system integrates a video player, several workflow history user interfaces, a video-workflow synchronization component, and a API document linker [18].

2 VT-REVOLUTION SYSTEM

2.1 Tutorial Authoring System

Synchronous Video Recording & Workflow History Logging To create an interactive programming video tutorial, a tutorial author can use a system that installs a screen-capturing tool (e.g., Snagit) and the ActivitySpace tool [3]. Screen-capturing tools usually support shortcut keys (for example, Snagit uses Shift+F9 and Shift+F10) to start and stop screen capturing. The ActivitySpace tool can be configured to monitor such key events. Once the ActivitySpace tool detects that the tutorial author starts (or stops) the video recording using the screen-capturing tool, it will start (or stop) the synchronous logging of the tutorial author's workflow history during the programming tutorial.

As the author is interacting with an application, ActivitySpace logs a time series of low-level HCI action records. Each action record has a time stamp down to millisecond precision. An action record is composed of event type and basic window information collected

Table 1: HCI Actions being Logged in IDE Editors and Views

UI Events	Type	When to Log	Content to Log
Loss focus	Edit	Key inputs occur	Component type, Window title File content, File display name
	View	–-	Component type View content, View name
Gain focus	Edit	–-	Component type, Window title File content, File display name
	View	–-	Component type View content, View name
Mouse click	Edit	Key inputs occur	Component type, Window title File content, File display name

using OS Window APIs, and focused UI component information that the application exposes to the operating system through accessibility APIs. The current prototype of tutorial authoring system configures ActivitySpace to log four types HCI actions in IDE editors and views as summarized in Table 1.

Workflow Operation Abstraction Considering the logging latency of HCI actions, the ActivitySpace tool just logs the action records during tutorial authoring. However, these low-level action records cannot intuitively reflect the developer's programming operations at higher-level of abstraction. Therefore, once the tutorial author stops the video recording and workflow history logging, our tutorial authoring system will abstract the logged time-series of low-level action records into a time series of high-level workflow operations. The current system prototype abstracts four categories of workflow operations pertinent to programming tasks: *open file* and *switch file*, *inspect exception*, *add and delete code elements*, and *edit text content*. Table 2 summarizes the heuristics for abstracting these workflow operations from low-level action records.

2.2 Tutorial Watching System

Figure 2 shows the screenshots of workflow history user interfaces in *VT-Revolution*. The design of these workflow history UIs allows tutorial watchers to easily explore workflow history, interact with tutorial content, and access API documents.

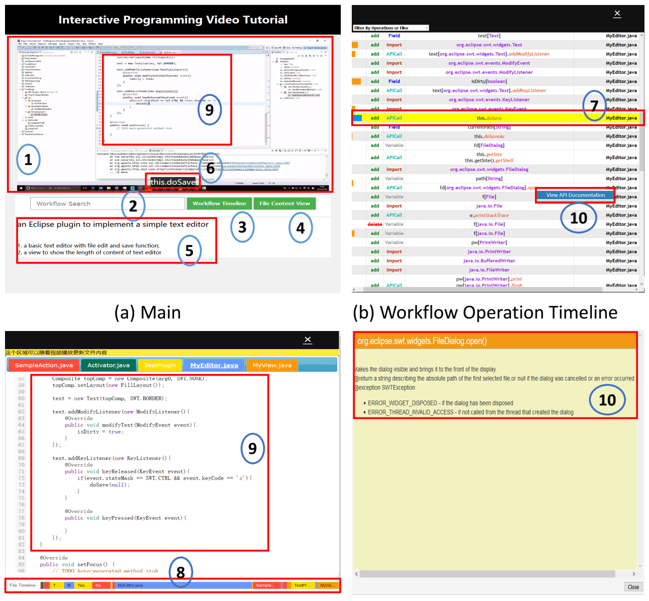
Video Caption: This feature allows tutorial watchers to easily note "*what the tutorial author does at this moment*" in the tutorial video. The video caption, which is automatically generated by the system based on workflow operations, highlights tutorial author's actions and code-element changes in the video content that is currently playing.

Workflow Operation Timeline: Workflow operation timeline provides an overview of "*when the tutorial author does what to which file*" during a programming tutorial. Workflow operations are shown chronologically from top down in the workflow operation timeline. Each row represents an operation incluing the time span of the operation (a horizontal bar proportional to the time span), the operation type, and the involved file(s). For add/delete-code-elements operations, the rows display the involved code elements. For inspect-exception and edit-text-content operations, the rows display a link which can be clicked to view details of exception or text content changes in a pop-up view.

Workflow operation timeline allows tutorial watchers to *search and navigate tutorial content by workflow operations.* Tutorial watchers can filter the workflow operations by operation type or the involved file(s). The tutorial video timeline and the workflow operation timeline are synchronized. As the tutorial video is playing or tutorial watchers navigate the tutorial video timeline, the workflow

Table 2: Workflow Operation Abstraction

Operation Category	Operation Type	Notion	Abstraction Heuristics
File	Open	FileOpen $< t_i, name >$	*File display name at the time t_i does not appear in the set of file display names from all the action records till the time t_{i-1}*
	Switch	FileSwitch $< t_i, origin, target >$	*The file display name at the time t_i is different from the file display name at the time t_{i-1}*
Exception	Inspect	Inspect $< t_{i-1}, t_i, exception >$	*The console output view has the focus from the time t_{i-1} to t_i, and the view content contains string "exception"*
Code Element	Add	Add $< t_{i-1}, t_i, type, info >$	*Unmatched AST node in the AST at the time t_i, compared with the AST at the time t_{i-1}*
	Delete	Delete $< t_{i-1}, t_i, type, info >$	*Unmatched AST node in the AST at the time t_{i-1}, compared with the AST at the time t_i*
Text content	Edit	Edit $< t_{i-1}, t_i, file, change >$	*Text content differences in the non-source text file from the time t_{i-1} to t_i*



an Eclipse plugin to implement a simple text editor

1. a basic text editor with file edit and save function;
2. a view to show the length of content of text editor

```
        Composite topComp = new Composite(arg0, SWT.NONE);
        topComp.setLayout(new FillLayout());

        text = new Text(topComp, SWT.BORDER);

        text.addModifyListener(new ModifyListener() {
            @Override
            public void modifyText(ModifyEvent event) {
                isDirty = true;
            }
        });

        text.addKeyListener(new KeyListener() {
            @Override
            public void keyReleased(KeyEvent event) {
                if(event.stateMask == SWT.CTRL && event.keyCode == 's') {
                    doSave(null);
                }
            }

            @Override
            public void keyPressed(KeyEvent event) {

            }
        });
    }

    @Override
    public void setFocus() {
        // TODO Auto-generated method stub
```

org.eclipse.swt.widgets.FileDialog.open()

Makes the dialog visible and brings it to the front of the display.
@return a string describing the absolute path of the first selected file,or null if the dialog was cancelled or an error occurred
@exception SWTException

- ERROR_WIDGET_DISPOSED - if the dialog has been disposed
- ERROR_THREAD_INVALID_ACCESS - if not called from the thread that created the dialog

(a) Main (b) Workflow Operation Timeline (c) File Content View (d) API Document Linking

Figure 2: Screenshots of *VT-Revolution*. (1) Video player. (2) Search Workflow Operations. (3) Show Workflow Operation Timeline. (4) Show File Content View. (5) A programming task description in tutorial. (6) Video Caption: workflow operation at this moment. (7) The highlighted workflow operation is synchronous with the video playing. (8) The file timeline: when to work on which file and time spent. (9) File content is synchronous with the video playing. (10) Right click an operation in operation timeline to access API document.

operation involved in the current video content will be highlighted in yellow color. The video watcher can navigate the tutorial video by double-clicking an operation in the workflow operation timeline.

File Content View File content view allows tutorial watchers to view "*all the content that the tutorial author already created to a file till to the current time of video playing*" during a programming tutorial. Tutorial watchers can show/hide this view by clicking "Show/Hide File Content View" button. The files that have been opened till the current time of video playing are displayed in a tabbed view. Each tab is annotated with the display name of a file. The focused file and its content is synchronized with the tutorial video playing. As the tutorial video is playing or tutorial watchers navigate the video, the focused file in the current video content will be underlined in the file content view. As changes are made to the focused file in the tutorial video, the content of the focused file will be updated automatically in the file content view. Although only a part of the focused file is visible in the current video content, tutorial watchers can switch between files and view file contents in the file content view just like in the IDE, without the need to navigate the video to the time when that content is visible.

File content view uses a timeline of file to provide an overview of "*when the tutorial author works on which file and the time spent*" during a programming tutorial . Each file in the timeline is represented by a distinct color. The same color is used in the corresponding file tab. The time spent on a file is proportional to a horizontal bar on the timeline. The horizontal bar is annotated with the corresponding file display name. This file timeline and the tutorial video timeline are synchronized. Combined with video and file content synchronization, the file timeline allows tutorial watchers to *navigate tutorial content based on the time when the tutorial author works on a particular file and creates particular file content.*

Search Workflow Operations This feature allows tutorial watchers to *find workflow operations that involve code elements they are interested in.* Tutorial watchers enter a keyword in the search box. The system currently performs a simple substring match between the entered keyword and the name attribute of code elements involved in workflow operations. It returns a list of work operations that involve the matched code elements in a chronological order. Tutorial watchers can double-click an operation in the results list to *navigate the tutorial video to the start time of the double-clicked operation.*

Accessing API Documentation While tutorial watchers inspect workflow operations in workflow operation timeline or workflow operation search results, or view code content in file content view, they can select a code element and request complementary learning resources. The current prototype supports the access to the official API documentation of the selected code element. We adopt the approach proposed in the Live API documentation tool [18] for linking the selected code element to its relevant API document.

3 EVALUATION

We conduct a usability study to evaluate the proposed system, *VT-Revolution*. We first record three Java programming tutorial videos using a screen-capture tool and ActivitySpace. Then we design a questionnaire for each tutorial video and invite 90 junior Java developers in an IT company to complete these questionnaires using *VT-Revolution*. 90 developers are divided into six groups: three experimental groups and three corresponding control groups. Each group has 15 participants. The participants in an experimental group and the corresponding control group. Finally, we analyze the results of the questionnaires and illustrate how *VT-Revolution* to

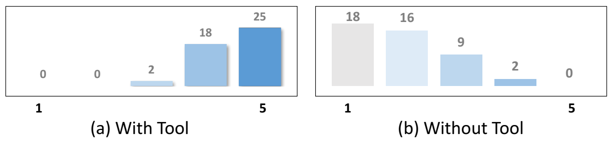


Figure 3: The Overall Satisfaction Score

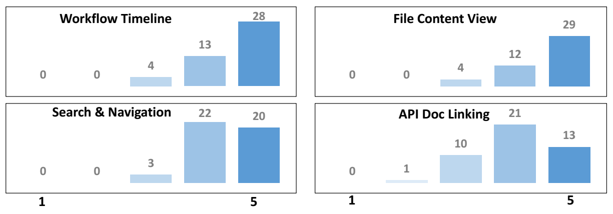


Figure 4: The Score of Different Functions of *VT-Revolution*

improve practitioners the efficiency of learning information from tutorial videos. We put all the experiment material in *VT-Revolution* prototype website http://baolingfeng.xyz:8080/VTRevolution/.

Figure 3 shows the overall satisfaction score rated by the participants with or without *VT-Revolution*. We can see that most of the participants using *VT-Revolution* have positive satisfaction scores while the majority of the participants using regular video player have negative satisfactory score. We also study the usefulness scores of different features rated by the participants in the experimental groups (see Figure 4). The majority of users rate these features useful or very useful. Therefore, We think *VT-Revolution* can help developers learn programming knowledge in video tutorials.

4 RELATED WORK

Video tutorials have been proved to be an effective medium for learning, for example, by providing user-guided experience [7] and encouraging learners to explore and learn at their own pace [13]. However, video tutorials also have some drawbacks, for example, navigation issues within long video tutorials [10], and it's hard for users who lack of overall understanding of recorded workflow to keep up with the pace of the instructions. Past research has shown that navigation and understanding of workflows in video tutorials can be aided by providing operation history and timeline-based browsing interactions [9, 14]. These tools are designed for drawing applications and graphical design software. Our system incorporates operation history and timeline interactions specially designed for software data, such as file switching, code-element changes.

Researchers have investigated how to create effective video for education [11, 19]. Some tools [8, 9] use operation histories to help users understand the captured workflow. However, these approaches require special recording tools and instrumentation of target applications. In contrast, our system does not have such requirements due to the adoption of the ActivitySpace framework.

5 CONCLUSION AND FUTURE WORK

We propose an interactive programming video tutorial system, *VT-Revolution*, to enhance the learning experience of programming video tutorials. By linking workflow history with video playback, tutorial watchers can obtain a high-level overview of workflow and file content, and directly navigate to parts of interest in their learning of video tutorials. Our user study confirms that our system can help developers learn video tutorials more efficiently, and lead to more satisfactory learning experience. The current prototype of *VT-Revolution* only supports several workflow operation abstractions in IDEs. In the future, we will extend it by supporting more applications and more workflow operations. ing resources: code and workflow.

ACKNOWLEDGMENT

This is a demonstration paper accompanying with our full research paper [4]. This research was partially supported by the National Key Research and Development Program of China (2018YFB1003904) and NSFC Program (No. 61602403).

REFERENCES

[1] Lingfeng Bao, Jing Li, Zhenchang Xing, Xinyu Wang, Xin Xia, and Bo Zhou. 2017. Extracting and analyzing time-series HCI data from screen-captured task videos. *Empirical Software Engineering* 22, 1 (2017), 134–174.
[2] Lingfeng Bao, Jing Li, Zhenchang Xing, Xinyu Wang, and Bo Zhou. 2015. Reverse engineering time-series interaction data from screen-captured videos. In *Proc. SANER*. IEEE, 399–408.
[3] Lingfeng Bao, Zhenchang Xing, Xinyu Wang, and Bo Zhou. 2015. Tracking and Analyzing Cross-Cutting Activities in Developers' Daily Work (N). In *Proc. ASE*. IEEE, 277–282.
[4] Lingfeng Bao, Zhenchang Xing, Xin Xia, and David Lo. 2018. VT-Revolution: Interactive Programming Video Tutorial Authoring and Watching System. *TSE* (2018).
[5] Lingfeng Bao, Zhenchang Xing, Xin Xia, David Lo, and Ahmed E. Hassan. 2018. Inference of development activities from interaction with uninstrumented applications. *Empirical Software Engineering* 23, 3 (2018), 1313–1351.
[6] Lingfeng Bao, Deheng Ye, Zhenchang Xing, Xin Xia, and Xinyu Wang. 2015. ActivitySpace: a remembrance framework to support interapplication information needs. In *Proc. ASE*. IEEE, 864–869.
[7] Peter Duffy. 2008. Engaging the YouTube Google-eyed generation: Strategies for using Web 2.0 in teaching and learning. *The Electronic Journal of e-Learning* 6, 2 (2008), 119–130.
[8] Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. 2011. Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning. In *Proc. UIST*. ACM, 373–382.
[9] Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2010. Chronicle: capture, exploration, and playback of document workflow histories. In *Proc. UIST*. ACM, 143–152.
[10] Susan M Harrison. 1995. A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. In *Proc. CHI*. ACM Press/Addison-Wesley Publishing Co., 82–89.
[11] Colin Lankshear and Michele Knobel. 2010. DIY Media: A contextual background and some contemporary themes. *DIY media: Creating, sharing and learning with new technologies. New York: Peter Lang* (2010), 1–21.
[12] Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. 2015. Code, camera, action: How software developers document and share program knowledge using YouTube. In *Proc. ICPC*. IEEE Press, 104–114.
[13] DF Mullamphy, PJ Higgins, SR Belward, and LM Ward. 2010. To screencast or not to screencast. *Anziam Journal* 51 (2010), C446–C460.
[14] Toshio Nakamura and Takeo Igarashi. 2008. An application-independent system for visualizing user operation history. In *Proc. UIST*. ACM, 23–32.
[15] Susan Palmiter, Jay Elkerton, and Patricia Baggett. 1991. Animated demonstrations vs written instructions for learning procedural tasks: a preliminary investigation. *International Journal of Man-Machine Studies* 34, 5 (1991), 687–701.
[16] Catherine Plaisant and Ben Shneiderman. 2005. Show me! Guidelines for producing recorded demonstrations. In *Proc. VL/HCC*. IEEE, 171–178.
[17] Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Massimiliano Di Penta, Rocco Oliveto, Mir Hasan, Barbara Russo, Sonia Haiduc, and Michele Lanza. 2016. Too long; didn't watch!: extracting relevant fragments from software development video tutorials. In *Proc. ICSE*. ACM, 261–272.
[18] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API documentation. In *Proc. ICSE*. ACM, 643–652.
[19] William Sugar, Abbie Brown, and Kenneth Luterbach. 2010. Examining the anatomy of a screencast: Uncovering common elements and instructional strategies. *The International Review of Research in Open and Distributed Learning* 11, 3 (2010), 1–20.
[20] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E Hassan, and Shanping Li. 2017. Measuring program comprehension: A large-scale field study with professionals. *TSE* (2017).