VT-Revolution: Interactive Programming Video Tutorial Authoring and Watching System

Lingfeng Bao, Zhenchang Xing, Xin Xia, David Lo

Abstract—Procedural knowledge describes actions and manipulations that are carried out to complete programming tasks. An effective way to document procedural knowledge is programming video tutorials. Unlike text-based software artifacts and tutorials that can be effectively searched and linked using information retrieval techniques, the streaming nature of programming videos limits the ways to explore the captured workflows and interact with files, code and program output in the videos. Existing solutions to adding interactive workflow and elements to programming videos have a dilemma between the level of desired interaction and the efforts required for authoring tutorials. In this work, we tackle this dilemma by designing and building a programming video tutorial authoring system that leverages operating system level instrumentation to log workflow history while tutorial authors are creating programming videos, and the corresponding tutorial watching system that enhances the learning experience of video tutorials by providing programming-specific workflow history and timeline-based browsing interactions. Our tutorial authoring system does not incur any additional burden on tutorial authors to make programming videos interactive. Given a programming video accompanied by synchronously-logged workflow history, our tutorial watching system allows tutorial watchers to freely explore the captured workflows and interact with files, code and program output in the tutorial. We conduct a user study of 135 developers to evaluate the design and effectiveness of our system in helping developers learn programming knowledge in video tutorials.

Index Terms—Program Comprehension, Human-Computer Interaction, Workflow

1 Introduction

To accomplish a program task, developers need not only *conceptual knowledge*, i.e., knowledge about concepts and APIs involved in the task, but also *procedural knowledge*, i.e., actions and manipulations that apply conceptual knowledge in the task. Software artifacts, such as code created in a task, records factual knowledge for software development. Procedural knowledge is usually documented in text-based or video-based programming tutorials that explain the flow of actions or events in the course of task completion. Interacting with software artifacts and tutorials and harnessing the knowledge in them has become an integral part of software development.

In term of authoring, it is often easier to record a 5-minutes programming video using screen-capturing tools like Snagit[1] than it is to take the necessary screenshots and write up an easy to follow text-based tutorial [33]. Furthermore, programming videos offer live and interactive experience which is absent in text-based tutorials. Programming videos can serve as a reasonable approximation of watching a developer's live coding practice. Seeing a developer's coding in action, for example, how changes are made to the source code step-by-step and how errors occur and are being fixed, can be more valuable than text-based tutorials [26].

In spite of these advantages, programming videos suffer some important drawbacks. Unlike text-based software artifacts and tutorials that can be effectively searched and linked using information retrieval techniques, the streaming nature of programming videos, i.e., a stream of screen-captured images, limits the ways that video watchers can interact with the tutorial workflow and content and harness the knowledge in the video. These limitations include:

- *Lack of a high-level overview of the workflow*. A programming task often involves several files and many changes made to the files step-by-step. Because programming videos lack a good summarization of the workflow history, it is often difficult for a video watcher to get an overview of the task completion process, e.g., when and how to modify which file, or what content the tutorial author already added to a file up till now.
- *No effective navigation support of workflow and tutorial content*. Unlike in the IDE where developers can freely switch between files to view their content, video watchers can only view the file content that is currently visible at a specific time. Furthermore, as video content is non-searchable images, it is difficult to navigate to a point in the video which contains a specific information of interest, for example, the time when a specific API call is added to a particular source file, or the time when a specific error occurs.
- *Inconvenience in linking to supplementary resources*. A programming video may involve APIs that a tutorial watcher is unfamiliar with, but the tutorial author does not explain in detail. As such, tutorial watchers often

- Lingfeng Bao is with the College of Computer Science and Technology, Zhejiang University, China.
E-mail: lingfengbao@zju.edu.cn
- Zhenchang Xing is with the Research School of Computer Science, Australian National University, Australia.
E-mail: zhenchang.Xing@anu.edu.au
- Xin Xia is with the Faculty of Information Technology, Monash University, Australia, and College of Computer Science and Technology, Zhejiang University, China.
E-mail: xxia@zju.edu.cn
- David Lo is with the School of Information Systems, Singapore Management University, Singapore.
E-mail: davidlo@smu.edu.sg

[1]https://www.techsmith.com/snagit.html

need supplementary resources (e.g., API documentation) to assist their learning of the tutorial content. As video content cannot be directly copy-pasted and used for search, tutorial watchers have to memorize the information in the video and use it to find relevant resources manually.

These limitations in interacting with tutorial workflow and content can lead to misunderstandings of the video content [20], difficulties in keeping up with the pace of the tutorial and reduced knowledge retention [32]. Studies show that adding interactive elements or workflows in videos can effectively enhance the learning experience of video tutorials [40], [24], [28], [31]. Simply put, the goal of our work is to make programming video tutorials interactive so that tutorial watchers can freely explore the workflow of a programming task in the video (e.g., jumping steps, filtering actions, navigating to a specific action or event), and interact with files, code and program output in the video in a similar way to the IDE interaction.

Multimedia tutorial authoring tools (e.g. *HyperCard*[2], *Adobe Authorware*[3], *Adobe Director*[4]) allow authors to create interactive workflows and elements in videos to enhance the video watcher's interaction with the tutorial. However, the interaction is limited to only the author-defined workflow and elements, which can be inefficient for programming tasks. Furthermore, creating multimedia video tutorials incur a significant burden on tutorial authors. Alternatively, some approaches [35], [5] use Optical Character Recognition (OCR) to convert video content into text so that video watchers can interact with tutorial content via OCRed text. However, OCR-based approaches cannot recognize actions in the videos, such as switching views, minor editing of file content, and thus cannot reliably recover the workflow history of the tutorial. That is, although OCR-based approaches do not increase the efforts for tutorial authoring, they provide only limited interaction with tutorial content.

To tackle the dilemma between the desirable interactive programming video tutorials and the undesirable burden on tutorial authors, two correlated design challenges must be addressed. First, what interaction designs would be effective for programming videos, considering the characteristics of programming tasks and data? Second, which level and what kinds of workflow history data is required for supporting programming-specific interactive designs, and how can we unobtrusively collect the required workflow history data to obviate the burden on tutorial authors?

In this paper, we design a set of programming-specific workflow history and timeline-based browsing interactions that allow programming video watchers to freely explore the workflow of a programming task and interact with files, code and program output in the video tutorial. To obviate the burden on tutorial authors, we leverage operating-system (OS) level instrumentation to log the tutorial author's low-level Human-Computer Interaction (HCI) data while he is interacting with software development tools. Sufficient details of HCI data are collected so that it can be abstracted into intuitive programming operations to support efficient interactions with the workflow and artifacts in the programming videos.

We recruit 135 developers from two IT companies in a user study to evaluate our *VT-Revolution* system. The user study involves three programming video tutorials with different functions and complexities. Participants are asked to use our *VT-Revolution* system, an OCR baseline prototype that assist video watching with OCRed-text based search, or just regular video player to watch video tutorials, and then answer the questions that cover four types of information related to the programming tasks in the tutorials. We analyze the correctness of answers, the time taken, and the satisfaction score of the learning experience between the participants using different tools. Our results show that developers using our *VT-Revolution* system can learn programming tutorials better and faster, and are more satisfied with the learning experience, compared with those using the OCR prototype and video player.

This paper makes the following contributions:

- It presents *VT-Revolution*, and to our best knowledge, we are the first to build a practical interactive programming video tutorial authoring and watching system. with special consideration of software data characteristics in system design, to capture procedural knowledge during programming tasks and allow developers to freely interact with procedural knowledge
- It presents the results of a user study with 135 developers to evaluate our *VT-Revolution* system, and the results show *VT-Revolution* improves the baseline approaches substantially.

Paper Structure: The remainder of the paper is structured as follows. Section 3 and 4 present the implementation of *VT-Revolution* system. Section 5 presents the procedure and results of the user study. Section 6 discusses several directions for further exploration. Section 7 reviews related work. Section 8 concludes the paper.

2 System Overview

As shown in Figure 1, our *VT-Revolution* system contains a tutorial authoring system and a tutorial watching system. The tutorial authoring system does not require any special tutorial authoring tools like *Adobe Director*. Neither does it require the instrumentation of screen-capturing tools or software development tools (e.g., IDEs). Instead, our tutorial authoring system integrates a regular screen-capturing tool (e.g., Snagit), the ActivitySpace framework [8], [6], and a workflow operation abstraction component. While the tutorial author is interacting with software development tools to create a programming tutorial, a screen-capturing tool records a video of the programming task, and, at the same time, the ActivitySpace framework synchronously logs the workflow history during the task. The workflow history logged by ActivitySpace is a time series of the tutorial author's human-computer interaction (HCI) actions and relevant contents in software development tools. The file management component extracts various pieces of information of files that the user has opened from action records, e.g name, content, and timestamp. It also keeps all versions of such files. The workflow operation abstraction component abstracts low-level HCI actions and contents into high-

[2]http://hypercard.org/
[3]http://www.adobe.com/products/authorware/
[4]http://www.adobe.com/products/director.html

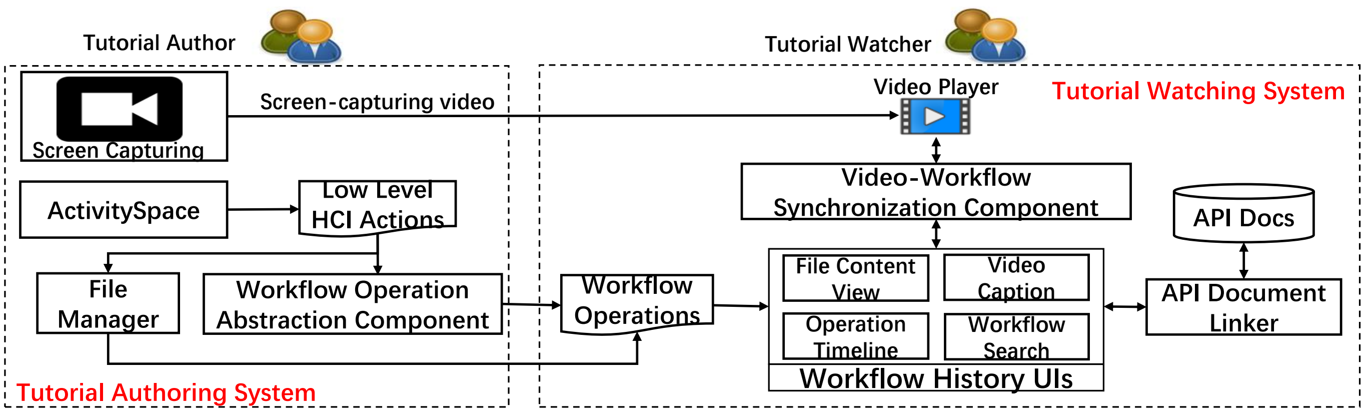


Fig. 1: The Framework of the *VT-Revolution* system

level workflow operations pertinent to programming tasks, such as open/switch files, inspect exceptions, add/delete code elements.

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 an API document linker [39]. The video player plays programming video. Workflow history user interfaces (see feature summary in Table 4 and screenshots in Figure 1) enhance programming video with automatic workflow-operation caption, and support timeline-based visualization, exploration and search of workflow operations and file contents. The video-workflow synchronization component supports synchronous video-operation and video-file navigation of tutorial content. The API document linker allows tutorial watchers to conveniently access API documents for code elements involved in the programming tutorial from within workflow history user interfaces.

3 Tutorial Authoring System

Our tutorial authoring system does not require any special interactive tutorial authoring tools like *Adobe Director*[4]. Instead, it consists of a synchronous video recording and workflow history logging component and a workflow operation abstraction component. The design challenge for the tutorial authoring system is how to log the tutorial author's activity at a low enough level to obviate the need for instrumenting specific software development tools, while ensuring that sufficient activity details are logged to abstract high-level programming operations and reconstruct workflow history.

3.1 Synchronous Video Recording & Workflow History Logging

This component consists of a screen-capturing tool (e.g., Snagit[1]) and the ActivitySpace tool [8], [6]. While the tutorial author is interacting with software development tools to create a programming tutorial, a screen-capturing tool records a screen-captured video of the programming task, and, at the same time, the ActivitySpace tool synchronously logs the workflow history during the task.

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

The ActivitySpace tool developed by Bao *et al.* [8], [6] is a HCI data collection framework that is specially designed for tracking a developer's interactions with a wide range of software tools commonly used in software development, such as IDEs, web browsers, text editors. Screen-capturing tools usually support shortcut keys (for example, Snagit uses Shift+F9 and Shift+F10) to start and stop screen capturing. We configure the ActivitySpace tool 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.

ActivitySpace does not require any application-specific instrumentation to log the developer's actions within an application. Instead, it incorporates OS window APIs, accessibility APIs and simple image-matching technique to monitor the developers' actions in a software tool and extract action-related information. When creating a programming video tutorial, the tutorial author usually spends most of the time in the IDE to demonstrate how to complete the programming task. Therefore, as the first proof-of-concept of our interactive programming video tutorial system, we log only the tutorial author's actions in the IDE. Considering the working environment of our study participants, we deploy ActivitySpace to collect HCI data in the Eclipse IDE on Windows. However, the feasibility study of ActivitySpace [8] shows that it can be deployed in other IDEs and on other operating systems.

As the developer is interacting with the IDE, ActivitySpace logs a time series of low-level HCI action records.

Each action record has a time stamp at millisecond precision. An action record is composed of event type and window information collected using OS Window APIs and focused UI component information that the application exposes to the operating system through accessibility APIs.

We configure ACTIVITYSPACE to log three UI events and five types of HCI actions in IDE editors and views as summarized in Table 1. For example, when an UI component losses the UI focus, if the UI component is an *edit* component and some *key inputs occur* while the UI component has the UI focus, an action record is logged which consists of: UI component type, *window title* obtained by OS window API, *file content* obtained from the UI value of the UI component, and *file display name* obtained from the Parent UI Name of the UI component. Similarly, when other UI events of interest occur and the relevant logging conditions are satisfied, corresponding action records will be logged with relevant contents obtained by OS Window APIs and accessibility APIs. The design space for HCI data to log is huge. The four type of actions we currently log is based on the frequent actions in the empirical study of developers' work data [6], If needed, ACTIVITYSPACE can be configured to collect more actions like clicking button, inspecting project hierarchy, etc.

The tutorial authoring system has a file manager component that extracts the file information including name, content, and timestamp from action records and keeps all the versions of files that the user has opened. These file information is used for workflow operation generation (e.g., AST generation, file content comparison), and the file content is automatically synchronized with the tutorial videos during tutorial watching (File Content View, see Section 4.1.3).

3.2 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 in the workflow history. Therefore, once the tutorial author stops the video recording and workflow history logging, the workflow operation abstraction component will abstract the logged low-level action records into a time series of workflow operations. Table 2 summarizes the heuristics for abstracting these workflow operations from low-level action records.

Based on our field study on developers' activities [7], the current system prototype abstracts four categories of workflow operations pertinent to programming tasks: *open/switch file*, *inspect exception*, *add/delete code elements*, and *edit text content*. Although the number of inspecting exception actions in video tutorials is much less than other types of actions (e.g.,, Ponzanelli et al. found that about 5% of video tutorials are related to bug/error fixing [35].), we believe that even when the tutorial author plans the tutorial very well, unexpected issues do happen. Furthermore, fixing exceptions or other runtime errors would be challenging if some less experienced developers run into them during the learning of the tutorial.

3.2.1 Open/Switch File

The workflow operation abstraction component maintains a set of distinct file display names from all the action records till the time point t_{i-1}. If the file display name of the action record at the time t_i does not appear in this list of file display names, the abstraction component generates an operation $FileOpen < t_i, name >$ where $name$ is the corresponding file display name at the time t_i. If the component type of the two consecutive action records at the time t_{i-1} and t_i are both edit component and the file display name of the two action records are different, the abstraction component generates an operation $FileSwitch < t_i, origin, target >$ where $origin$ and $target$ are file display name at the time t_{i-1} and t_i respectively.

TABLE 3: Details of Add/Delete Code Element Operations

ASTNode	Type	Info
Import statement	Import	*Package name in the import statement*
Field declaration	FieldDeclare	*Field name, Field datatype name*
Variable declaration	VarDeclare	*Variable name, Variable datatype name*
Method call	MethodCall	*Method identifier, Object and its datatype on which a method is called*

3.2.2 Inspect Exception

If the component type of the two consecutive action records at the time t_{i-1} and t_i are both view component, the view names are the console view name of the IDE[5], and the view content contains string "exception", this indicates that console output view has the focus in between the time t_{i-1} and t_i and the view content displays some programming exceptions. Therefore, the abstraction component generates an operation $InspectException < t_{i-1}, t_i, exception >$ where $exception$ is the view content of the action record. Our approach can be easily extended to collect "Examine Compile Errors" actions. We do do not include this type of action in our current prototype because compile errors in programming tutorials are usually minor and easy to fix.

3.2.3 Edit File

If the component type of the two consecutive action records at the time t_{i-1} and t_i are both edit component and the file display name of the two action records are the same, the abstraction component attempts to abstract changes made to the file in between t_{i-1} and t_i. The system is currently configured to distinguish two types of files: source file and non-source text file. This is done by matching the file extension extracted from the window title of the action record with the a set of source file extensions as input parameters to the system. For example, files with ".java" extension are considered as source files, while file with extension like ".xml", ".properties" or ".txt" are considered as non-source text files. Note that file display name of an UI component may not contain this information. For example, for the Eclipse plugin configuration file *plugin.xml*, the file display name of the editor is the plugin name, not the file name.

• **Edit text content**. For non-source text files, the abstraction component uses a text differencing tool [6] to compute the

[5]Our system can be configured to check console view name of different IDEs.

[6]https://code.google.com/p/java-diff-utils/

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*

TABLE 4: Tutorial Watching System Feature Summary

	Workflow Overview	Search & Navigation	Access API Doc
Video Caption	• *What the tutorial author does at this moment*	—	—
Workflow Operation TimeLine	• *When the tutorial author does what to which file*	• *Filter operations by operation type or file* • *Synchronous video-operation navigation*	• *Code element in the selected operation*
File Content View	• *All the content of a file till the current time of tutorial video* • *When the tutorial author works on which file and the time spent*	• *Switching between files* • *Synchronous video-file navigation*	• *Code elements in current file content*
Search Workflow Operations	—	• *All operations involve searched code elements* • *Synchronous video-operation navigation*	• *Code element in the returned operation*

differences between the file content of the two consecutive action records at t_{i-1} and t_i. It generates edit-text-content operation $Edit < t_{i-1}, t_i, file, change >$ where $file$ is the file display name of the action records and $change$ is the text differences of the file content in between t_{i-1} and t_i.

• **Add/delete code elements**. For source files, the abstraction component attempts to detect various types of code elements being *added* and/or *deleted* in between t_{i-1} and t_i, because code-element changes correspond better to the developers' intuition about code changes than simple text changes [42], [17]. To detect code-element changes, the abstraction component first attempts to parse the file content of an action record using an appropriate parser, for example a Java parser for Java source files [7]. Note that source files may contain incomplete code fragments during a programming tutorial and thus may not be parsable.

If the file content of the two consecutive action records are both parsable, the abstraction component compares the two obtained ASTs to detect changes to code elements of interest. As the code changes between the two consecutive time points in a programming tutorial are usually minor, we implement a simple AST differencing algorithm to compare the two ASTs. The algorithm recursively compares the two ASTs from the root node down level by level. The two nodes (one from each AST) are considered as a match if they have the same set of attributes (e.g., node type, node name). After tree differencing, we obtain a list of unmatched AST nodes in each AST. Based on the unmatched AST nodes in each AST, the abstraction component generates code element change operations $Delete < t_{i-1}, t_i, type, info >$ and $Add < t_{i-1}, t_i, type, info >$. The current prototype focuses on four types of code elements as summarized in Table 3: import statement, field declaration, variable declaration, and method call, because they are usually code elements that a code search engine returns, such as Codebase[8], Krugle[9]. To extend the system, other types of code elements can be extracted from the AST differencing results in the same way.

[7]https://github.com/javaparser/javaparser

4 Tutorial Watching System

Table 4 summarizes the key features of our tutorial watching system. Figure 2 shows the screenshots of current prototype. Our tutorial watching system takes as input a screen-captured programming video and a synchronously-logged time-series of workflow operations. The design challenge for the tutorial watching system is how to seamlessly incorporate programming video and workflow history data to allow tutorial watchers to easily find the needed information in the video tutorial.

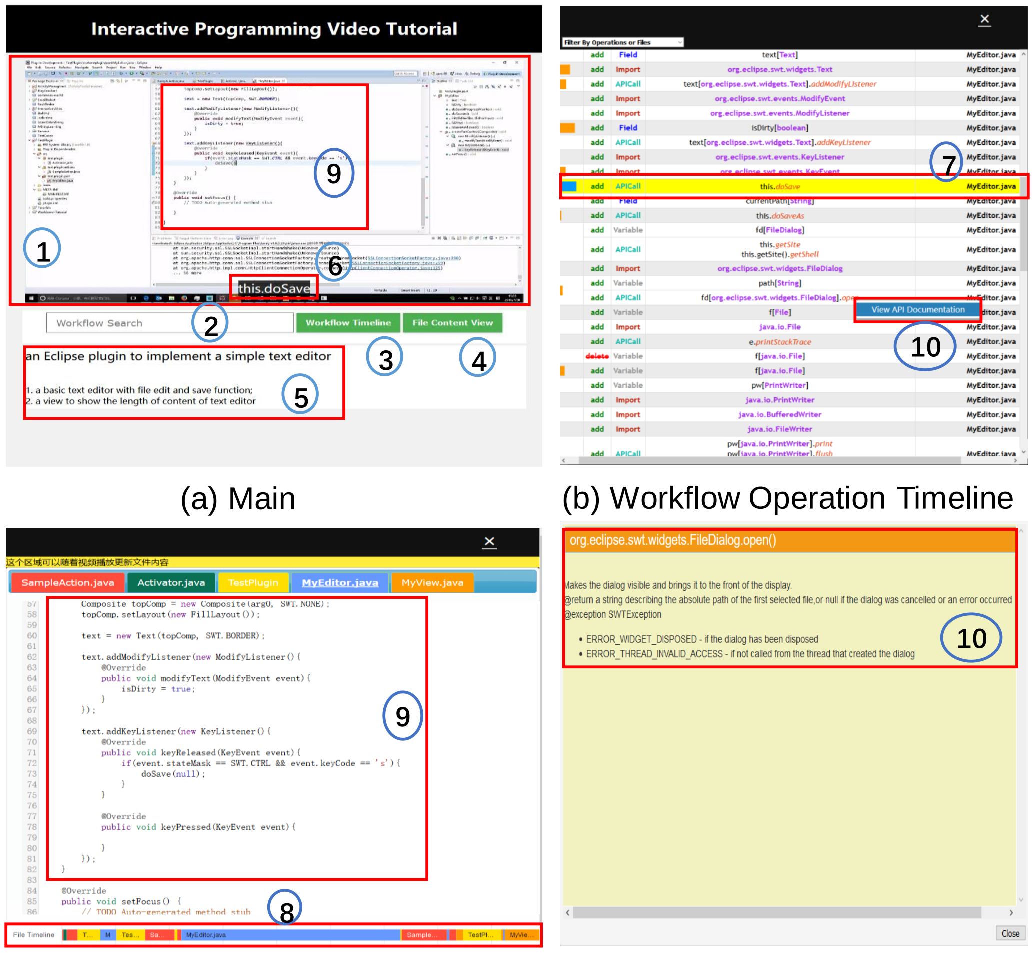
4.1 Workflow History User Interfaces

Workflow history user interfaces enhance programming video with automatic workflow-operation caption, and support timeline-based visualization, exploration and search of workflow operations and file contents. These UIs use a video-workflow synchronization component for synchronous navigation of video content and workflow operations (or files).

4.1.1 Video Caption

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

[8]http://www.codase.com/
[9]http://www.krugle.com/



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

(a) Main

add	Field	text[Text]	MyEditor.java
add	Import	org.eclipse.swt.widgets.Text	MyEditor.java
add	APICall	text[org.eclipse.swt.widgets.Text].addModifyListener	MyEditor.java
add	Import	org.eclipse.swt.events.ModifyEvent	MyEditor.java
add	Import	org.eclipse.swt.events.ModifyListener	MyEditor.java
add	Field	isDirty[boolean]	MyEditor.java
add	APICall	text[org.eclipse.swt.widgets.Text].addKeyListener	MyEditor.java
add	Import	org.eclipse.swt.events.KeyListener	MyEditor.java
add	Import	org.eclipse.swt.events.KeyEvent	MyEditor.java
add	APICall	this.doSave	MyEditor.java
add	Field	currentPath[String]	MyEditor.java
add	APICall	this.doSaveAs	MyEditor.java
add	Variable	fd[FileDialog]	MyEditor.java
add	APICall	this.getSite this.getSite().getShell	MyEditor.java
add	Import	org.eclipse.swt.widgets.FileDialog	MyEditor.java
add	Variable	path[String]	MyEditor.java
add	APICall	fd[org.eclipse.swt.widgets.FileDialog].ope	ditor.java
add	Variable	f[File]	ditor.java
add	Import	java.io.File	MyEditor.java
add	APICall	e.printStackTrace	MyEditor.java
delete	Variable	f[java.io.File]	MyEditor.java
add	Variable	f[java.io.File]	MyEditor.java
add	Variable	pw[PrintWriter]	MyEditor.java
add	Import	java.io.PrintWriter	MyEditor.java
add	Import	java.io.BufferedWriter	MyEditor.java
add	Import	java.io.FileWriter	MyEditor.java
add	APICall	pw[java.io.PrintWriter].print pw[java.io.PrintWriter].flush	MyEditor.java

(b) Workflow Operation Timeline

```
        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
```

(c) File Content View

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

(d) API Document Linking

Fig. 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.

4.1.2 Workflow Operation Timeline

Workflow operation timeline provides an overview of "*when the tutorial author does what to which file*" during a programming tutorial. Tutorial watchers can show this timeline by clicking "Workflow Operation Timeline" button below the video player. It shows chronologically all workflow operations from top down. Each row represents an operation, and it shows: 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 operation involved in the current video content will be highlighted in yellow color. Double-clicking an operation in the workflow operation timeline highlights the double-clicked operation as the current operation and navigates the tutorial video to the start time of the double-clicked operation. At any time, tutorial watchers can view the operations before and after the current operation in the workflow operation timeline without the need to navigate the video to the time when an operation occurs.

4.1.3 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*.

4.1.4 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 (see Table 3) 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*.

4.2 Accessing API Documentation

While tutorial watchers inspect workflow operations in workflow operation timeline or workflow operation search results, view code content in file content view, they can select a code element and request supplementary resources. Our current prototype supports the access of the official API documents of the selected code element.

We adopt the approach proposed in the Live API documentation tool [39] for linking the selected code element to its relevant API document. We build the knowledge base of the official API documents by extracting the API Javadocs from the source code of open source projects. The knowledge base currently includes the official API documents for JDK, Eclipse, and Java mail library. For a select code element, the API document linker selects a set of initial candidate APIs in the knowledge base whose name matches the name of the selected code elements, and then it uses the context-based disambiguation mechanism proposed in the Live API documentation tool to determine the most appropriate API for the selected code element. This context-based disambiguation mechanism essentially filters irrelevant candidate APIs by other code elements co-occurring in the same file of the selected code element. This mechanism has been shown to be effective in identifying a unique API for a code element in a code snippet [39]. When several candidate APIs are still left after context-based disambiguation, the system displays the API documents of all of them for user selection.

5 Experiment

In this section, we conduct a controlled experiment with 135 developers who do not use Java as their main programming language. We use our tutorial authoring system to produce three interactive Java programming video tutorials. For each tutorial, we design a questionnaire based on the programming task of the tutorial. Our goal is to evaluate how well developers can answer programming questions related to the programming tutorials with or without *VT-Revolution*.

5.1 Research Questions

The research questions guiding our study are:

RQ1 How well and efficiently does our *VT-Revolution* system help developers search relevant information in video tutorials, compared with developers using the OCR prototype and regular video player?

Motivation. Our *VT-Revolution* system enhances the video tutorial learning experience by providing visualization, exploration and search of the captured workflow history and tutorial content. We would like to investigate whether *VT-Revolution* can make a significant difference in helping participants understand the important knowledge and information in a video tutorial better and faster.

RQ2 Are the participants using *VT-Revolution* more satisfied with the learning experience of the video tutorials than those using the OCR prototype and regular video player?

Motivation. *VT-Revolution* offers an interactive tutorial learning experience, which is completely different from just watching a video. We would like to investigate the user experience and the participants' subjective assessment on *VT-Revolution*, compared with that of the participants using the OCR prototype and regular video player.

RQ3 Which feature(s) of *VT-Revolution* are most useful?

Motivation. Our *VT-Revolution* system includes a rich set of interaction features for exploring workflow history and interacting with tutorial content (see Table 4). We would like to investigate the usefulness of these features and obtain some insights to improve *VT-Revolution*.

5.2 Programming Tutorials

Experimental tutorials are designed based on watching similar programming videos on the Internet. The first author who has five-years Java programming experience first creates three Java programming video tutorials. The second and third authors are university lecturers who teach programming courses. We follow their teaching experiences in authoring programming tutorials for teaching programming practices. Then, all authors collaboratively revise the tutorials to adjust step sequence, proceeding speed, and tutorial transcripts.

These three programming video tutorials can be found on our *VT-Revolution* website[10]. As our study participants are all from China, the first author speaks Chinese when recording these tutorials.

Table 5 shows the details of the three programming tutorials. The first column is the name of the tutorial, the

[10]http://192.243.114.62:8080/VTRevolution/

TABLE 5: Three Java Programming Video Tutorials Used in Experiment

Tutorial	Programming Task	LOC	#File	Duration [mm : ss]	Log (MB)	Video (MB)	#Frame (OCR)
email	*a simple program to send email:* *1. load email configuration from a properties file* *2. send a test email using Java mail library*	75	2	8:39	0.38	49.5	77
mysql	*a program to illustrate some MySql Database operations* *1. query a table using Statement* *2. query a table using PreparedStatement* *3. insert a row into table then return the auto increment id* *4. call a stored procedure*	175	1	11:06	0.97	82.0	372
plugin	*an Eclipse plugin:* *1. a basic text editor with file edit and save function* *2. a view to show the length of content in text editor*	309	5	19:19	1.21	173.0	490

second column summarizes the programming task in the tutorial, the third column is the total line of code (LOC) after finishing the programming task at the end of tutorial, the forth column is the number of files that are opened and/or modified in the tutorial, and the fifth column is the duration of tutorial.

To evaluate the effectiveness of our *VT-Revolution* system in a diverse setting, we design three programming tutorials with different functionalities and complexity. The *email* tutorial is the simplest among the three programming tasks. The *email* tutorial is the shortest tutorial (about 9 minutes), and the final program is the smallest (75 LOC). The *email* tutorial involves two files, a source file and a property file that stores email configurations to be loaded when the program starts. The *mysql* tutorial is of medium duration and complexity. It implements four kinds of SQL database operations. The final program in the *mysql* tutorial has 175 LOC. The *mysql* tutorial involves only one source file. The *plugin* tutorial is the most complex programming task among the three tutorials, which involves 4 source files and 1 plugin configuration file. The final program has 309 LOC, and the tutorial duration is almost 20 minutes. Both the *email* and the *mysql* tutorial require programming with a library. The *plugin* tutorial is different from the other two tutorials in that it requires knowledge of extending Eclipse framework, which is a more complex task than calling library APIs.

The 6th and 7th columns of Table 5 show the size of screen-captured videos and the size of workflow history log data. We can see that the size of log data is very small, because ACTIVITYSPACE logs only action, time stamp and text content (if any). For example, the size of the *email* tutorial video recorded by Snagit is ∼50MB (even when we compress the video, its size is still ∼24M), while the size of the workflow history log data is just 0.38M.

5.3 Baseline tools

In this study, we consider two baseline tools. The first baseline is just regular video player to watch programming videos. The second baseline is an OCR prototype that assists video watching with OCRed-text based search and navigation. We follow the CODETUBE approach [35] to build the OCR prototype. First, we filter similar frames for the three experimental tutorials. The 8th columns of Table 5 shows the number of frame after filtering similar frame. Second, we identify Java code using computer vision and OCR techniques. With this OCR prototype, users can enter a text query to search the video, and the prototype returns all frames that contain the input query. Then users can choose some returned frames to navigate the video tutorial.

TABLE 6: Statistics of the Projects with Participants

Project	Years	#Devloper	Pro.	#Participant
A	6	136	C#	40
B	4	90	C#	25
C	4	18	C#	12
D	3	48	C#	15
E	2	10	Python	4
F	4	28	Python	12
G	2	32	C/C++	12
H	6	68	C/C++	15

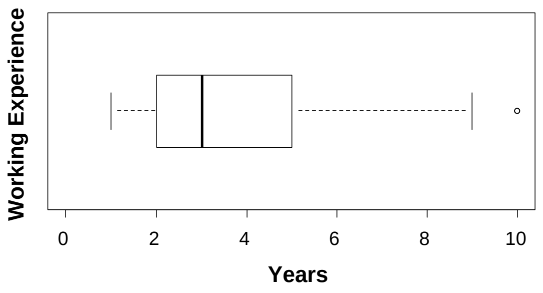


Fig. 3: The Working Experience of Participants

5.4 Participants

We recruit 135 professional developers from two IT companies in China, named Insigma Global Service [11] and Hengtian [12]. Insigma Global Service is an outsourcing company which has more than 500 employees, and it mainly does outsourcing projects for Chinese vendors (e.g., Chinese commercial banks, Alibaba, and Baidu). Hengtian is an outsourcing company which has more than 2,000 employees, and it mainly does outsourcing projects for US and European corporations (e.g., State Street Bank, Cisco, and Reuters). To recruit the developers, we obtain a list of projects in the two companies that do not use Java as main programming language (as part of their education, all developers have basic Java knowledge). We email the developers in the selected projects to seek volunteers for our study. 135 developers from 8 different projects are recruited. Table 6 shows the statistics of these 8 projects[13].

[11]http://www.insigmaservice.com/
[12]http://www.hengtiansoft.com/
[13]Due to the security policies in these two companies, we anonymize the project name.

The columns correspond to the name of project (Project), the time duration of the project (Years), the number of the developers (#Developer), the main programming language (Pro.), and the number of developers who participate our study (#Participant). These 8 projects have different time durations, sizes, and main programming languages. All the 135 developers have basic knowledge about Java programming but they do not use Java as main programming language in their work.

Then, we conduct a pre-study survey to ask participants the working experience and whether they have used Java to write the three program tasks (i.e., send an email, execute MySQL operations, create an eclipse plugin) in our study. We find that 91% (123) of participants said they had not written such kinds of programs, while the other participants said that they wrote one or two similar programs in their college days but they had forgotten the exact steps. As such, the 135 participants are not familiar with the programming tasks used in the experiment.

Figure 3 presents the working experience of the participants. The average working experience is 3.69 years. We divide these 135 developers into three groups based on their working experience[14]: *senior* (more than 5 years of professional experience), *middle* (3 to 5 years of professional experience) and *junior* (less than 3 years of professional years). Among the 135 participants, 26, 59, and 50 of them belong to *senior*, *middle*, and *junior* category, respectively. Due to developers' time constraints, we adopt between-subject design in our experiment. Based on the participants' years of professional experience, we divide the 135 developers into nine comparable groups: three groups that use *VT-Revolution* (experimental group), three groups that use the OCR prototype (control group 1), and three groups that use regular video player (control group 2). Each group has 15 participants and the ratio of *junior*, *middle*, and *senior* developers for each group is kept approximately the same as 1:2:2.

We deploy *VT-Revolution* and the baseline tools as web applications so that participants can access them via a web browser easily. We divide the whole experiment into three sessions. In each session, participants in three groups for one programming tutorial are required to complete the questionnaire questionnaire for the relevant programming tutorial (see Section 5.5). The participants may complete the tasks before the session ends. Depending on their groups, the participants use *VT-Revolution*, the OCR prototype or regular video player to watch the programming tutorial, and complete a questionnaire about programming knowledge in the watched tutorial.

5.5 Questionnaire Design

To evaluate how well participants learn programming knowledge in video tutorials, we design a questionnaire for each video tutorial. Each questionnaire has several questions, and each question is related to a knowledge point in the video tutorial. Some questions ask the participants to select one or more answers from multiple choices, while other questions ask the participants to fill in blank with relevant information. The first author develops standard answers to the questionnaires, which are further validated by the second and third author. A small pilot study with three developers (one for each tutorial) is conducted to test the suitability and difficulty of the tutorials and questionnaires. The complete questionnaires can be found in our *VT-Revolution* website[10].

[14]We decide the developer level based on the input from these two companies.

TABLE 8: The Number of Questions for Experimental Video Tutorials

	Total Question	API Usage	Workflow	Output	File Content
email	9	4	2	1	2
mysql	9	3	3	2	1
plugin	11	4	3	1	3

It is important to note that these questionnaires are not designed in favor of the *VT-Revolution* features. Instead, they are designed based on our programming experiences and a survey of two developers, with the goal to cover four categories of information that tutorial watchers may be interested in when learning a programming tutorial. Table 7 show example questions of different categories for the three video tutorials. Table 8 shows the number of questions of different categories for the three video tutorials.

1) API Usage: This category has two sub-categories. API usage (in tutorial) sub-category asks tutorial watchers to find the information about how an API explained in the tutorial is used in the programming task of the video tutorial. In contrast, API usage (API doc) sub-category contains questions about APIs that appear in the tutorial but the tutorial does not explain in detail. We give tips to participants that the answers to API usage (API doc) questions cannot be found in the tutorial.

2) Output: Output questions require tutorial watchers to find program output information in a video tutorial. The output can be normal results or exceptions after program execution. Program output information can help tutorial watchers identify important knowledge point in the tutorial. For example, seeing what program may cause an exception and how the exception can be fixed is very valuable for a novice developer.

3) File Content: File content questions require tutorial watchers to find the information about program configuration and the resulting code after certain programming actions.

4) Workflow: Different from locating a specific information in a video tutorial, workflow questions check tutorial watchers' understanding of the overall process of completing a task or subtask.

It is important to note that, to evaluate the participants' learning outcomes of a video tutorial, we do not ask developers to create a program from the video tutorial, because a developer can literally copy-paste code in the tutorial to make a working program, without understanding the task completion process, why it works, and what to do when something goes wrong. In contrast, our experimental questions explicitly test the developers's ability to search, explore and understand the tutorial to satisfy their information needs in learning, such as what are appropriate parameters for an API call, how to handle an exception, and what is the next step to complete the code.

TABLE 7: Example Questions of Different Categories for Experimental Video Tutorials

Category	Video	Question Examples
API Usage (in tutorial)	email	*A variable **session** of type **javax.mail.Session** is declared. How is this variable initialized?*
	mysql	*How to create a **Statement** instance in the method **executeStatement** of the class **DBImpl**?*
	plugin	*What is the Class of the first parameter in the API call **activePage.openEditor**, and what does the second parameter represent?*
API Usage (API Doc)	email	*If the API **java.mail.Message.setFrom** is called without any parameters, which attribute in the **Property** will be regarded as the "From" in the email? If no such attribute, what is the value of "From" in the email?*
	mysql	*When creating **Statement** object, how to obtain a **ResultSet** that is scrollable, updateable, and insensitive to updates by others?*
	plugin	*List the parameter list of all overloading APIs **org.eclipse.ui.IWorkbenchPage.openEditor**, except the one used in the video.*
Output	email	*In the video, what kinds of exceptions are thrown?*
	mysql	*What is the query result after completing and running the method **executePreparedStatement**?*
	plugin	*When running the Eclipse plugin, an **IllegalArgumentException** is thrown. To fix this problem, the value of the first argument of **activePage.openEditor** is changed from **null** to a new value. What is this new value?*
File Content	email	*Which attributes are configured in the property file?*
	mysql	*What is the name of output parameter of stored procedure?*
	plugin	*How many override methods in the class **MyEditor** are changed in the tutorial?*
Workflow	email	*What is the API call sequence on the variable **message** whose class is **javax.mail.Message**?*
	mysql	*Before you get a MySQL database connection instance, what step you must do?*
	plugin	*In the video, to create the UI of text editor, what is the code fragment in the method **createPartControl** of the class **MyEditor**?*

5.6 Experiment Procedure

Before the experiment, we give a short tutorial on the features of the *VT-Revolution* system to participants in the experimental group. The training focuses only on system features. We do not instruct the participants how they could use our system to answer questionnaires. For the control group using the OCR prototype, we introduce how to use the prototype to search and navigate the video tutorial. For the control group using regular video player, no training is needed.

At the beginning of an experiment session, a questionnaire web page is shown to the participants. Once the participants click the start button, a web page is opened in another browser window or tab. The participants in the three groups use the corresponding tool to watch the video tutorial. Participants can answer the questionnaire while watching the video tutorial. For API usage (API doc) questions, the groups using the OCR prototype and regular video player can use search engines to find the relevant API documentation. When the participants complete the questionnaire, they submit the questionnaire by clicking the submit button.

After submitting the questionnaire, the participants in the three groups are asked to rate an overall satisfaction score of the corresponding tool (*VT-Revolution*, the OCR prototype, and regular video player) they use for learning video tutorials. They can also write some suggestions or feedbacks to us. We use open card sorting [38] to group the comments from the participants. In the preparation phase, we create one card for each feedback comment. In the execution phase, we extract some key words from each comment, then match the key words to group the comments. Finally, we choose some representative comments from each group. One author and two graduate students jointly sort the card. For the group using *VT-Revolution*, we also ask the participants to rate the usefulness of the *VT-Revolution* features: workflow operation timeline, file content view, search and navigation, and API documentation linking. All of the scores rated by participants are 5-points likert scales (1 being the worst, 5 being the best).

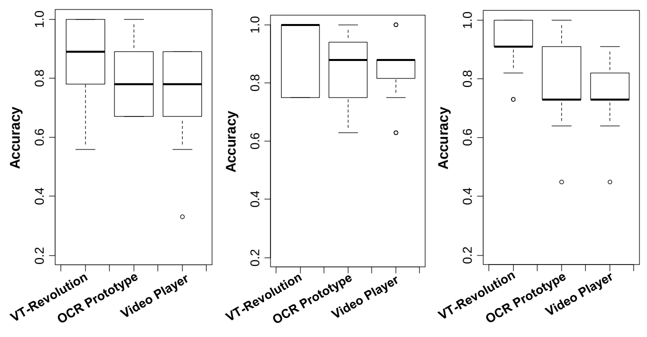
5.7 Experiment Results

In this section, we show the results of our experiment to answer three research questions.

5.7.1 RQ1

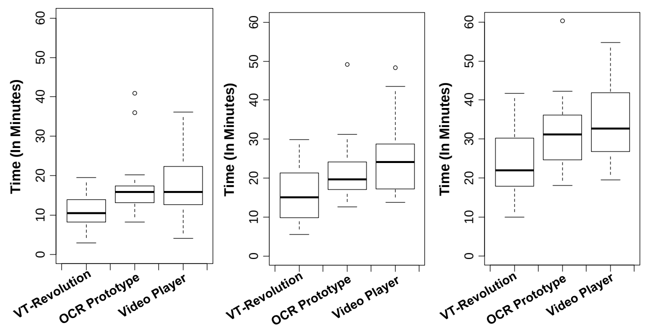
Approach. The accuracy of a participant's answers to the questions in the questionnaire and the time spent to complete the questionnaire can reflect how well and how efficient the participant understands a video tutorial. We define accuracy as the percentage of correct answers in a questionnaire. The higher the accuracy is, the better a participant understands the video tutorial. An answer to a question is marked as correct if it contains only the choice(s) or information as expected in the standard answer. The first author marks the participants' answers, which are validated by the second and third authors. To get the total time to complete a questionnaire, we collect the start time when the start button is clicked and the end time when the submit button is clicked. The shorter the completion time is, the more efficient a participant finds relevant information the video tutorial.

We use Wilcoxon Rank Sum test [41] to measure whether the difference of accuracy and time between experimental



(a) email (b) mysql (c) plugin

Fig. 4: The Accuracy of Answers to Questionnaires for the Three Experimental Video Tutorials



(a) email (b) mysql (c) plugin

Fig. 5: The Time of Completing Questionnaires for the Three Experimental Video Tutorials

group using *VT-Revolution* and control group using the OCR prototype (or regular video player) is statistically significant. p value < 0.05 in Wilcoxon Rank Sum test is considered as statistically significant. We compute Cliff's delta [11] [15], which is a non-parametric effect size measure that quantifies the amount of difference between the two groups.

Results. Figure 4 presents in box plots the accuracy of answers to the questionnaires for the three experimental video tutorials. The data of the three groups is labeled as *VT-Revolution*, *OCR* and *Video Player* respectively. We can see that the average accuracy of answers in the experimental groups using *VT-Revolution* is higher than that of the control groups using OCR prototype or regular video player (on average 0.89 *vs.* 0.80 and 0.74 for *email*, 0.94 *vs.* 0.85 and 0.84 for *mysql*, 0.91 *vs.* 0.78 and 0.75 for *plugin*). For the *mysql* tutorial, 11 out of the 15 participants in the experimental group using *VT-Revolution* answer the whole questionnaire correctly, while only 4 in the control group using the OCR prototype and 3 in the control group using regular video player achieve this. For the *email* and *plugin* tutorial, there are still some participants in the experimental groups who answer the whole questionnaire correctly, while only 1 participant in the control groups using the OCR prototype achieves this and none of the participants in the control groups using regular video player achieve this.

Figure 5 presents in box plots the time of completing questionnaire for the three experimental video tutorials. We can see that the average time of completing questionnaire in the experimental groups using *VT-Revolution* is less than that of the control groups using OCR prototype or regular video player (on average 11.17 *vs.* 17.77 and 17.16 minutes for *email*, 16.05 *vs.* 22.26 and 25.67 minutes for *mysql*, 23.95 *vs.* 32.01 and 34.81 minutes for *plugin*). An interesting observation is that the time spent by many participants in the experimental groups using *VT-Revolution* is actually less than the duration of the video tutorial. This indicates that participants using *VT-Revolution* may not need to watch the entire tutorial videos in order to find relevant information in the videos. This suggests that *VT-Revolution* can be very useful when developers are not interested in the entire tutorial but only some part(s) of the tutorial. In contrast, the participants using the OCR prototype and regular video player usually spend much longer time than the duration of the video tutorial.

Table 9 presents the *p-values* of Wilcoxon Rank Sum test with Bonferroni correction and the Cliff's deltas of the accuracy and time for the three experimental video tutorials. The *p-values* are all smaller than 0.05 which means that the differences between the experimental groups and the corresponding groups are statistically significant at the confidence level of 95%, except for the accuracy between the group using *VT-Revolution* and the group using the OCR prototype for the *mysql* tutorial, and the time between the group using *VT-Revolution* and the group using video player for the *email* tutorial. The differences are also substantial with the Cliff's deltas belonging to large effect size, except for the accuracy between the group using *VT-Revolution* and the group using the OCR prototype for the *mysql* tutorial.

Comparing the accuracy and time of the control groups using the OCR prototype and regular video player, we can see that using the OCR prototype can help people find relevant information faster and better than using just video player. But the improvement is much smaller, compared with using *VT-Revolution*. Furthermore, although the difference of using *VT-Revolution* and the OCR prototype is not significant in the *mysql* tutorial involving only one source file, *VT-Revolution* can help people find relevant information significantly faster and better in more complex tutorials involving several files.

Feedbacks we receive from the participants using *VT-Revolution* provide some evidence that how our tool helps them answer the questionnaires better and faster. Some examples are given below:

- *"Some questions that are related to source code in the video are very easy to answer using this tool, since I can view all code files at any timestamp of the video tutorial."*
- *"Using this tool, I need not to watch the video tutorial from start to end, I just use the keyword in the question to search workflow history."*
- *"I can understand some APIs through viewing the documents directly using this tool, and need not to open another web page to search online."*

Feedbacks from the participants using the OCR prototype suggest that OCRed text can help search and navigate the video, but it is often not accurate. some examples are given below:

- *"I use some keywords, e.g. 'exception', and find the target location in the video very quickly."*

[15]Cliff defines a delta of less than 0.147, between 0.147 to 0.33, between 0.33 and 0.474, and above 0.474 as negligible, small, medium, and large effect size, respectively.

TABLE 9: The P-values and Cliff's deltas of Accuracy and Time

		OCR Prototype			Video Player		
		email	**mysql**	**plugin**	**email**	**mysql**	**plugin**
accuracy	**p-value**	0.0420	0.1834	0.0138	0.0264	0.0488	0.0008
	Cliff's delta	0.4267 (M)	0.2711 (S)	0.5156 (L)	0.5511 (L)	0.4844 (L)	0.7689 (L)
time	**p-value**	0.0210	0.0488	0.0296	0.0556	0.0296	0.0198
	Cliff's delta	-0.4933 (L)	-0.4267 (M)	-0.4667 (M)	-0.5000 (L)	-0.5467 (L)	-0.5733 (L)

S, M, and L are small, medium, and large effective size, respectively.

- *"In many times, I use a keyword from the question but do not find any results. In this situation, this tool doesn't work."*
- *"Some keywords return too many target locations. Checking the returned locations one by one is very difficult and time consuming."*

Feedbacks from the participants using regular video player are generally not good. Some typical difficulties they are faced with are given below:

- *"Watching a 20-minutes video tutorial in which the programming task is not well known to me continuously is very boring. But I have to watch it very carefully to answer the questions."*
- *"I remember I saw something somewhere relevant to the question, but i cannot recall the precise part of the video for the relevant information. Finding it in the video is really painful."*
- *"As I do not know Eclipse APIs well, remembering the interface and API name to be searched is a kind of mental challenge."*

5.7.2 RQ2

Approach. We compare the overall satisfaction score of the tools used by the experimental and control groups for learning video tutorials. We use Wilcoxon Rank Sum test [41] to measure whether there is a significant difference between the scores rated by the participants using *VT-Revolution* and those using the OCR prototype (or regular video player) We compute Cliff's delta [11] to quantify the effect size of the difference between the two groups.

Results. Figure 6 shows the overall satisfaction score rated by the participants using *VT-Revolution*, the OCR prototype and regular video player. We can see that almost all the participants (43 out of 45) using *VT-Revolution* have positive satisfaction scores (4 or 5), and only 2 participants are neutral. For the OCR prototype, The number of participants who have positive satisfaction scores (12 out of 45) is close to that of participants who have neutral (16 out of 45) and negative satisfactory scores (17 out of 45). The majority (34 out of 45) of the participants using regular video player have negative satisfactory score (1 or 2). The average satisfaction score rated by the participants using our *VT-Revolution* is 4.51, while the average satisfaction score rated by the participants using the OCR prototype and regular video player is only 2.83 and 1.89, respectively. The $p-value$ between the groups using *VT-Revolution* and the groups using the OCR prototype is 1.184e-11, and The $p-value$ between the groups using *VT-Revolution* and the groups using regular video player is 7.45E-16. This indicates that the satisfaction differences are statistically significant at the confidence level of 99%. The Cliff's delta is 0.79 and 0.97 respectively, which corresponds to a large effect size.

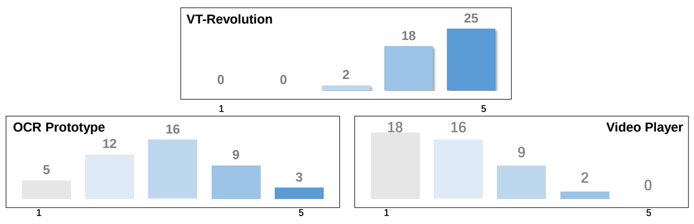


Fig. 6: The Overall Satisfaction Score

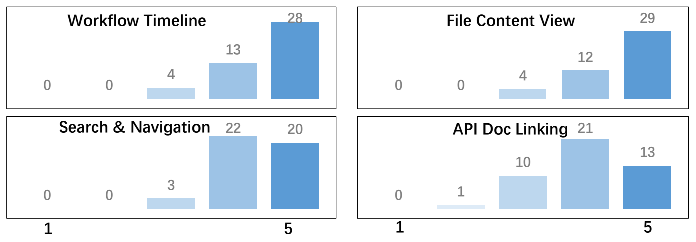


Fig. 7: The Score of Different Functions of *VT-Revolution*

Most feedbacks from the participants using *VT-Revolution* are positive, except two participants complaining loading time is very long. The reason is that they use non-common 360 Browser which has some issues with our current web prototype. Examples of positives feedbacks are:

- *"This tool is very interesting. I can clearly know what the video author does at any time. It's very useful to help me understand the video tutorial."*
- *"The code in text format is more familiar to me than the code in video. I can copy the code fragment from the video tutorial using this tool. Very cool!"*
- *"The prototype is designed very well. I can navigate the video easily and find what I want."*

The participants using OCR prototype appreciates the usefulness of OCRed text but propose some suggestions to improve the OCR prototype, for example:

- *"I can use this tool to navigate the video tutorial, but for some questions in the questionnaire that require the context and programming process, that's not enough. I have to spend more time to look into the tutorial."*
- *"If the search results can show more information, e.g. what's the content of the returned location about, this will help locate the target frame more faster, especially when too many results are returned."*

The participants using regular video player have many complaints, for example:

- *"I always have no patience with watching the whole video tutorial, but watching video discontinuously usually make me miss some important things."*

- 👎 *"Even though I can locate the information in the video, I often need to watch this fragment of the video repeatedly so that I can find out what really happen."*

From the above feedbacks, we can see that *VT-Revolution* can be helpful to solve some typical problems that are encountered by the participants using just video player.

5.7.3 RQ3

Approach. We study the usefulness scores of different features rated by the participants using *VT-Revolution*. We also review the feedbacks from these participants on these features.

Results. From Figure 7, we can see that the average scores of all *VR-Revolution* features are above 4 (i.e., useful). The users like the most workflow operation timeline and file content view. The majority of users rate these two features are useful or very useful. For search and navigation feature, about half of the participants rate it as useful, while the other half rate it as very useful. The API document linking feature receives the lowest usefulness score.

Some comments from the users illustrate why they think these features are useful:

- 👍 **Workflow operation timeline:** *"I can know the whole workflow more clearly using this timeline and use it to navigate video more easily."*
- 👍 **File content view:** *"File content view gives me an overview of the program in the video, and it is easy to know the code change by comparing the code content at two different times."*
- 👍 **Search and navigation:** *"I like the synchronization between the video and the workflow. I can easily find the needed information and jump to that video part."*
- 👍 **API Doc linking:** *"I do not know the usage of many classes and APIs in the video tutorial since I never write Eclipse plugin programs. I can understand the video better using API documentation, just like what I can do in the IDE"*

Some participants also provide good suggestions to help us improve *VT-Revolution* in the future, for example:

- ☞ *"I can only view the API documentation, but sometimes it's not enough. It could be better to link to other resources (e.g. Stack Overflow) or use the selected code as a query to search the Web."*
- ☞ *"This tool is very good now, but it will be much better if it can analyze and abstract some good practices of tutorial authors such as frequently used shortcut keys."*
- ☞ *"If this tool can be applied to large amount of existing online video tutorials, it will be very valuable. "*

5.8 Threats to Validity

Threats to internal validity refer to factors internal to our studies that could have influenced our experiment results:

1) The quality of video tutorials: The first author who creates the experimental tutorials is very experienced in the involved programming tasks. To ensure the quality, all authors collaboratively review the created video tutorials and the first author makes necessary revisions to step sequence, proceeding speed, and tutorial transcripts. However, we are not aware of a methodical approach to generating the tutorials in the literature. In the future, we might develop some methodical approach to generating programming tutorials based on more experiences of creating video tutorials and more feedbacks from professional developers. Furthermore, the simple programming tasks of the first two video tutorials might be a threat to validity since participants with basic Java knowledge might answer the questions without watching video tutorials. However, all questions except for those that belong to API usage (API doc) category require participants to find relevant information in the video tutorials. Moreover, it is difficult to answer the questions of API usage (API doc) category without looking up API documentation. We also confirm that most of participants do not know the three program tasks well in the pre-study study.

2) The implementation of OCR prototype: We have followed the approach of CODETUBE [35]. But we use another OCR tool ABBYY FineReader[16], which is a professional OCR software that has better OCR accuracy than the open source OCR tool TESSERACTOCR. We use a smaller dissimilarity threshold (0.001) as the threshold of CODETUBE (0.1) leaves only several frames after filtering similar frames. In a pilot study of the OCR prototype, all participants complain that too few frames lead to incomplete understanding of the workflow and inaccurate content navigation.

3) The appropriateness of the questionnaires: All authors design the questionnaires together with the goal to cover different categories of knowledge that developers could be interested in the programming tutorials. However, there might be some biases that favor *VT-Revolution* in the questionnaires. For instance, it is very difficult to answer some questions that belong to "File Content" category using the baseline tools since users have to inspect the video frame by frame to identify the file content change. We recruit three developers to test-run the experiment and make necessary revisions of questionnaires based on their feedbacks on the suitability and difficulty of tutorials and questionnaires. Our test-runs ensure that not only the participants using *VT-Revolution* but also the participants using the baseline tools can answer all the questions in the questionnaires.

4) The equivalence of experiment and control groups: We conduct a pre-study survey of participants' years of programming experience and familiarity with similar programming tasks and make the best effort to divide them into comparable groups. 5) the Hawthorne effect [1], namely that participants change their behavior because they know they are being watched: Since *VT-Revolution* is a new tool, which could change participants' behavior, we could never fully exclude the Hawthorne effect, like many studies in software engineering research [9], [2].

Threats to external validity refer to the generalizability of the results in the experiment. Although our experiment uses only three video tutorials, we carefully design the three programming tasks with different functions and complexities. The video tutorials we create have typical workflows like those online programming videos. To improve the generalizability of the results, we invest significant effort to recruit 135 developers from two IT companies in our experiment.

[16]https://www.abbyy.com/en-us/finereader/

6 Discussion

Based on our own experiences in designing *VT-Revolution*, and comments made by participants in our experiment, we see several directions for further exploration:

Working environment as a tutorial system: ActivitySpace [8], [6] aims to support self-retrospect of a developer's workflow history. In this work, we show that this workflow history can also become a powerful tool for finding relevant information in video tutorials. Imagine a scenario where the work flow of a senior developer on a task can be recorded and shared among team members. This would help junior developers improve their procedural knowledge about how they could better carry out a task. To make effective use of workflow history, advanced analysis of workflow data is required. For example, we can infer different kinds of tasks from a developer's workflow history, which is called task boundary identification [36]. This could help developers manage their work more efficiently and find best practices in their work.

Bridging conceptual and procedural knowledge in software engineering: Code sharing and reuse is an established practice. Unfortunately, the workflow by which developers carry out to generate the code is lost. Online programming videos capture the workflows of coding, but code in videos cannot be easily explored and reused. Our *VT-Revolution* system bridges these two complementary programming knowledge: code to be developed and how to develop it. This could lead to a new way of knowledge archiving and sharing in which workflows can be freely explored and, at the same time, code can be easily reused. Furthermore, *VT-Revolution* can be used to enhance software engineering education. For example, students can use *VT-Revolution* to record videos and workflows when they are doing a programming task. Then teachers can give comments on task completion process rather than focusing on only end results.

Making existing video tutorials interactive: Our system showcases the design opportunities that are introduced by having workflow data associated with videos. While it shows promise, the vast collection of existing online video tutorials does not possess such workflow data. Our study shows that OCRed text offers only limited help in searching and navigating videos. Recent years have seen the advance of computer-vision techniques, such as Prefab [12], Waken [3] and Sikuli [43], which can reverse-engineer workflow data from videos. Novel interaction design of *VT-Revolution*, in combination with the OCRed text and the reverse-engineered workflow data, could transform the learning of existing video tutorials into an interactive mode which is impossible for the video data alone. Furthermore, the recorded workflow data in *VT-Revolution* and the reverse-engineered workflow data can be used to link *VT-Revolution*-style programming tutorials with existing video tutorials. The two types of workflow data could also be used to optimize video search, which can help developers find target video tutorials faster.

7 Related Work

7.1 Video Tutorials

Video tutorials have been proved to be an effective medium for learning, for example, by providing user-guided experience [14] and encouraging learners to explore and learn at their own pace [29], [15]. A recent study by MacLeod *et al.* [26] on programming videos on YouTube shows that video tutorials are effective in providing an introduction to a technology and demonstrating how a piece of software can be developed within an IDE. A developer survey by Ponzanelli *et al.* [35] shows that many developers use video tutorial regularly.

In spite of their effectiveness for learning, video tutorials have some drawbacks. One major drawback is navigation issues within long video tutorials which could lead to misunderstandings of the content [20]. Additionally, users may be unable to keep up with the pace of the instructions due to the lack of overall understanding of recorded workflow, which could lead to reduced knowledge retention [32]. 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 [30], [18], [27], [34], [3]. Existing tools are designed for drawing applications and graphical design software. Our system is specially designed for software data and programming tasks.

Ponzanelli *et al.* [35] use Optical Character Recognition (OCR) to extract text from videos, which enables developers to query video content easily. Bao *et al.* [5] propose a computer-vision based video scraping technique to automatically extract time-series interaction data from programming videos. Computer-vision techniques have limitations in time cost and quality of extracted data. In contrast, our system builds on the ActivitySpace framework that collects precise workflow data in software tools.

7.2 Interactive Tutorial Authoring

Researchers find that interactive components or activities for video viewers are very effective for learning [40], [24], [28], [31]. Many tools [18], [3], [16] use operation histories to help users understand the captured workflow. These approaches require additional efforts to instrument target applications. In contrast, our system requires no additional instrumentation of IDEs and can be easily extended to instrument other applications (e.g., web browsers, text editors) due to the adoption of the ActivitySpace framework. Multimedia tutorial authoring tools, such as HyperCard[2], Adobe Director[4], allow users to create interactive videos. However, it can be very demanding to create programming tutorials using such tools, due to the flexibility of workflows to be captured and the amount of information (e.g., APIs) to be annotated. In contrast, our system does not incur any additional burden on tutorial authors, besides recording a tutorial video as usual. A programming tutorial video with logged workflow created by our system can be regarded as an interactive video. It is possible to editing recorded video and logged workflow in a similar way to editing an interactive video in interactive tutorial design tools like Adobe Authorware and Adobe Director. However, a specific video and workflow editing tool needs to be developed, which is out of the scope of this work.

7.3 Tracking User Interaction Data.

Tracking user's interaction data within IDEs can help improve developers' performance. For example, Mylyn listens to Eclipse IDE selection and view services to help manage the task in the Eclipse workbench [21]. Reverb recommends previously visited web pages that pertain to the code visible in the developers' editor [37]. Li *et al.* propose the amAssist tool which monitors developers' coding activities to recommend relevant web resources [25]. These approaches require to instrument IDEs to log user's activities [10], [23], [19], [13]. To obviate the need for application-specific instrumentation, our system uses the ActivitySpace framework [8], [6] which logs low-level HCI data using standard OS Windows APIs and accessibility APIs which are commonly provided by modern operating systems [4].

Some researchers propose several computer vision based techniques to extract actions from video tutorials, e.g. Pause-and-play [34], Waken [3]. The actions identified by these techniques are usually based on a set of icons in an application. However, when watching programming video tutorials, tutorial watchers often focus on the process of making a program work, e.g., code changes, API usage, debugging, etc. These computer vision based techniques can help watchers understand how to use a GUI application by extracting the action list from video tutorials but they cannot extract the kind of process knowledge in programming video tutorials. For example, Pause-and-play [34] is demonstrated to work on only two applications, Google SketchUp and Adobe Photoshop; Waken [3] can extract only actions like cursor movement, clicked icons, etc. Furthermore, Waken's experiments show that it extracts only 14 cursor movements and 8 icons from 34 Google SketchUp video tutorials, which demonstrates the difficulty of using computer vision techniques to track user interaction data in video tutorials.

Extracting and summarizing process knowledge in video tutorials could help video watchers quickly scan, filter, and review video tutorial. For example, Tooscape [22] uses a video browsing interface with a storyboard summarization and an interactive timeline to support the exploration of how-to process knowledge in video tutorial, but it uses crowdsourcing to ask people to label and verify video segments into steps manually. In contrast, our system tracks and abstracts user interaction data automatically.

8 Conclusion

We present a novel system that leverages automatically logged workflow history to enhance the interactive learning experience of programming video tutorials. By linking workflow history with video playback, tutorial watchers can obtain a high-level overview of workflow and relevant code and program output, and directly navigate to parts of the video that they are interested in, without having to watching the entire video. Our user study shows that our system can help developers find relevant information in video tutorials better and faster, and lead to more satisfactory learning experience. Participants' feedbacks suggest that our system can help solve many problems that are commonly encountered by watching the videos using just a video player or assisting video watching with only OCRed-text based search. Our current prototype VT-Revolution can abstract multiple actions including open/switch file, inspect exception, add/delete code elements, and edit text content from programming video tutorials. In the future, we consider adding more actions (e.g., examine compile error, run program) into our system. This work also inspires some future directions for software engineering practices, for example, how to transform working environment into an engaging tutorial system, and how to bridge conceptual and procedural knowledge in software engineering.

References

[1] J. G. Adair. The hawthorne effect: A reconsideration of the methodological artifact. *Journal of applied psychology*, 69(2):334, 1984.

[2] V. Augustine, P. Francis, X. Qu, D. Shepherd, W. Snipes, C. Braunlich, and T. Fritz. A field study on fostering structural navigation with prodet. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 229–238. IEEE, 2015.

[3] N. Banovic, T. Grossman, J. Matejka, and G. Fitzmaurice. Waken: reverse engineering usage information and interface structure from software videos. In *Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST)*, pages 83–92. ACM, 2012.

[4] L. Bao, J. Li, Z. Xing, X. Wang, X. Xia, and B. Zhou. Extracting and analyzing time-series hci data from screen-captured task videos. *Empirical Software Engineering*, pages 1–41, 2016.

[5] L. Bao, J. Li, Z. Xing, X. Wang, and B. Zhou. Reverse engineering time-series interaction data from screen-captured videos. In *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 399–408. IEEE, 2015.

[6] L. Bao, Z. Xing, X. Wang, and B. Zhou. Tracking and analyzing cross-cutting activities in developers' daily work (n). In *Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 277–282. IEEE, 2015.

[7] L. Bao, Z. Xing, X. Xia, D. Lo, and A. E. Hassan. Inference of development activities from interaction with uninstrumented applications. *Empirical Software Engineering*, pages 1–39, 2017.

[8] L. Bao, D. Ye, Z. Xing, X. Xia, and X. Wang. Activityspace: a remembrance framework to support interapplication information needs. In *Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 864–869. IEEE, 2015.

[9] M. Beller, G. Gousios, and A. Zaidman. How (much) do developers test? In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 559–562. IEEE, 2015.

[10] T.-H. Chang, T. Yeh, and R. Miller. Associating the visual representation of user interfaces with their internal structures and metadata. In *Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST)*, pages 245–256, 2011.

[11] N. Cliff. *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[12] M. Dixon and J. Fogarty. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 1525–1534. ACM, 2010.

[13] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. TaskTracer: a desktop environment to support multi-tasking knowledge workers. In *Proceedings of the 10th international conference on Intelligent user interfaces (IUI)*, page 75, 2005.

[14] P. Duffy. Engaging the youtube google-eyed generation: Strategies for using web 2.0 in teaching and learning. *The Electronic Journal of e-Learning*, 6(2):119–130, 2008.

[15] I. Duncan, L. Yarwood-Ross, and C. Haigh. Youtube as a source of clinical skills education. *Nurse education today*, 33(12):1576–1580, 2013.

[16] J. Fernquist, T. Grossman, and G. Fitzmaurice. Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning. In *Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST)*, pages 373–382. ACM, 2011.

[17] B. Fluri, M. Wuersch, M. PInzger, and H. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*, 33(11):725–743, 2007.

[18] T. Grossman, J. Matejka, and G. Fitzmaurice. Chronicle: capture, exploration, and playback of document workflow histories. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology (UIST)*, pages 143–152. ACM, 2010.

[19] E. Harpstead, B. A. Myers, and V. Aleven. In search of learning: facilitating data analysis in educational games. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI)*, page 79, 2013.

[20] S. M. Harrison. A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)*, pages 82–89. ACM Press/Addison-Wesley Publishing Co., 1995.

[21] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for IDEs. In *Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168, 2005.

[22] J. Kim. Toolscape: enhancing the learning experience of how-to videos. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 2707–2712. ACM, 2013.

[23] J. H. Kim, D. V. Gunn, E. Schuh, B. C. Phillips, R. J. Pagulayan, and D. Wixon. Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI)*, pages 443–451, 2008.

[24] C. Lankshear and M. Knobel. Diy media: A contextual background and some contemporary themes. *DIY media: Creating, sharing and learning with new technologies. New York: Peter Lang*, pages 1–21, 2010.

[25] H. Li, X. Zhao, Z. Xing, L. Bao, X. Peng, D. Gao, and W. Zhao. amassist: In-ide ambient search of online programming resources. In *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 390–398. IEEE, 2015.

[26] L. MacLeod, M.-A. Storey, and A. Bergen. Code, camera, action: How software developers document and share program knowledge using youtube. In *Proceedings of 23rd IEEE International Conference on Program Comprehension (ICPC)*, pages 104–114. IEEE Press, 2015.

[27] J. Matejka, T. Grossman, and G. Fitzmaurice. Ambient help. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 2751–2760. ACM, 2011.

[28] S. Mohorovičić. Creation and use of screencasts in higher education. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 1293–1298. IEEE, 2012.

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

[30] T. Nakamura and T. Igarashi. An application-independent system for visualizing user operation history. In *Proceedings of the 21st annual ACM symposium on User interface software and technology (UIST)*, pages 23–32. ACM, 2008.

[31] J. Oud. Guidelines for effective online instruction using multimedia screencasts. *Reference Services Review*, 37(2):164–177, 2009.

[32] S. Palmiter, J. Elkerton, and P. Baggett. Animated demonstrations vs written instructions for learning procedural tasks: a preliminary investigation. *International Journal of Man-Machine Studies*, 34(5):687–701, 1991.

[33] C. Plaisant and B. Shneiderman. Show me! guidelines for producing recorded demonstrations. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 171–178. IEEE, 2005.

[34] S. Pongnumkul, M. Dontcheva, W. Li, J. Wang, L. Bourdev, S. Avidan, and M. F. Cohen. Pause-and-play: automatically linking screencast video tutorials with applications. In *Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST)*, pages 135–144. ACM, 2011.

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

[36] I. Safer and G. C. Murphy. Comparing episodic and semantic interfaces for task boundary identification. In *Proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research*, pages 229–243, 2007.

[37] N. Sawadsky, G. C. Murphy, and R. Jiresal. Reverb: Recommending code-related web pages. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, pages 812–821. IEEE, 2013.

[38] D. Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

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

[40] W. Sugar, A. Brown, and K. Luterbach. Examining the anatomy of a screencast: Uncovering common elements and instructional strategies. *The International Review of Research in Open and Distributed Learning*, 11(3):1–20, 2010.

[41] F. Wilcoxon and R. A. Wilcox. *Some rapid approximate statistical procedures*. Lederle Laboratories, 1964.

[42] Z. Xing and E. Stroulia. Umldiff: an algorithm for object-oriented design differencing. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (ASE)*, pages 54–65. ACM, 2005.

[43] T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: using gui screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology (CHI)*, pages 183–192. ACM, 2009.