Code, Camera, Action: How Software Developers Document and Share Program Knowledge Using YouTube

Laura MacLeod, Margaret-Anne Storey and Andreas Bergen
University of Victoria, Victoria, BC, Canada
{lmacleod, mstorey, andib}@uvic.ca

***Abstract*—Creating documentation is a challenging task in software engineering and most techniques involve the laborious and sometimes tedious job of writing text. This paper explores an alternative to traditional text-based documentation, the screencast, which captures a developer's screen while they narrate how a program or software tool works. We conducted a study to investigate how developers produce and share developer-focused screencasts using the YouTube social platform. First, we identified and analyzed a set of development screencasts to determine how developers have adapted to the medium to meet the demands of development-related documentation needs. We also explored the techniques and strategies used for sharing software knowledge. Second, we interviewed screencast producers to understand their motivations for creating screencasts, and to uncover the perceived benefits and challenges in producing code-focused videos. Our findings reveal that video is a useful medium for communicating program knowledge between developers, and that developers build their online personas and reputation by sharing videos through social channels.**

I. Introduction

Software documentation is the castor oil of software development. —Gerald Weinberg, 1975

Much of current software development involves using or altering code authored by other developers. The existence of well-written documentation and relevant examples is often critical for other developers to understand code they did not write themselves. Yet efficiently and succinctly documenting software poses many challenges. Describing how a program or an application interface works often involves the tedious and laborious task of writing text, which many developers shun. Traditional documentation may be embedded in the program code itself (as comments), or it may be captured in documents such as help files and Wikis.

Developers are now leveraging social channels such as blogs, community portals, GitHub, Twitter and Stack Overflow to author and share documentation [1]. Sharing knowledge on social channels benefits the producer of the documentation as it helps build their online reputation and creates a network of like-minded developers [4].

Today, the diffusion of technology makes it easier for people to record and share videos. Some prefer video over text for communicating tacit knowledge, and we see videos being used for many knowledge sharing tasks. Developers have also begun using video to capture their screens and software development environments.

Screencasts are "movies of software" [5] that capture a developer's screen as they manipulate code for a specific example or goal. Screencasts can be used to demonstrate tool use and to present *how-to information* for using application program code, instances which can be difficult or tedious to describe in text. Screencasts are also frequently used for educational purposes. For software development, they are used in conjunction with written documentation and posted directly on popular project Websites (e.g., see the Ruby on Rails project[1]), or hosted and shared through social video channels.

In this paper, we investigate how and why developers create and share screencasts through YouTube. In the first phase of our study, we analyzed a broad set of development-related YouTube screencasts to determine how they are used for sharing programming knowledge. In a second phase, we interviewed screencast creators to gain insights into their motivations for creating and sharing these videos. The interviews revealed a variety of benefits and challenges screencast creators may experience when using and sharing video documentation. Our findings reveal that video is an efficient and useful medium for program documentation aimed at other developers, and that the social features of the hosting channels (our study focused on YouTube) play an important role in screencast development and distribution.

The main contributions of this paper are:

1) a first study of how and why screencasts are used to share development knowledge through YouTube;
2) an understanding of the types of knowledge developers share through screencasts;
3) a list of motivating factors for why developers create these screencasts and share them on YouTube; and
4) technical insights into how developers create screencasts and the challenges with using current video creation tools for documenting code.

In the remainder of this paper, we review the literature on the role of documentation within social software development and the literature on using YouTube for sharing knowledge in other domains, as well as outline our research methodology and research questions. We present the findings from an analysis of 20 developer screencasts hosted on YouTube and describe the findings from 10 interviews with developers who

[1]http://rubyonrails.org/screencasts/

have created screencasts. We relate our findings to the existing literature and then address the limitations of our study. Since this paper reports first findings on how YouTube is used by developers, we pose a number of questions for future research.

II. Background

Our study builds on three areas: the role of documentation in software engineering, the social nature of software documentation, and how YouTube is used by knowledge workers.

A. Software Documentation

Documentation is essential for the development and evolution of software. It describes many facets of the development process, including requirements, feature implementations, usage scenarios, code examples, and decisions made during development. Many studies have shown that documentation is needed to provide programmers with vital information. For example, Brooks discusses how documentation is used to help programmers understand a program [6]. Yet there are many challenges in producing software documentation, including managing large software documents and incorporating an appropriate level of detail for particular understanding needs [7]. Keeping documentation up to date may also be time consuming and costly for developers [7].

Today, most software documentation is text based and maybe embedded in books, source code comments, and bug tracking issues [8]. Wikis [8], question and answer sites [9], and technical articles [10] provide crowd-based ways to contribute knowledge in online communities. For software documentation, these platforms provide different benefits and limitations for developers (e.g., Wikis are easy to create and navigate, question and answer sites remain up to date longer, while blogs generate more fanfare [10]).

A drawback with text-based approaches for sharing developer knowledge is that they may not capture the tacit knowledge needed to understand workflows and processes. Tacit knowledge is best communicated through face-to-face interactions, verbal discussions, or more recently, through video [11]. Screencasts are becoming an increasingly popular way for developers to describe how others can navigate source code[2]. Such screencasts may capture video clips of a developer directly interacting with and manipulating source code, as well as showing the executed program. These screencasts provide visual and audio cues that go beyond the information portrayed by the static text of a program. Screencasts have the potential to supplement written documentation, and for some purposes, even surpass it. One area in which video may provide more support for developers is in the documentation of software execution and navigation. Previous work has found that executable products are often preferred over static documentation when providing development context [10]. Screencasts can capture software execution in addition to a developer's environment and insights.

[2]See https://news.ycombinator.com/item?id=8897389 for a recent discussion of the prevalence of screencasts for sharing developer knowledge

While our study investigated how video can be used for software documentation, we also sought to understand the social implications when video-based software documentation is shared through social channels. In the next subsection, we review existing research on the social nature of software documentation.

B. The Social Nature of Software Documentation

Today's developers work in "social ecosystems around content, media and developers" [1]. Social dissemination of documentation promotes collaboration with other individuals [12], helps find answers to questions [13], and increases developer awareness among communities [14].

Developer participation in social ecosystems leads to the emergence of online identities [14]. A developer's online identity becomes associated with their knowledge and skills [14], affording them informal membership in many communities of practice that build around members with shared interests and goals [15]. Peers use this information to evaluate other developers and to shape their interactions with them [16].

The transparency of social media not only helps developers establish identities, it also creates awareness about trends and cultural norms [14]. The ability to share new developments instantaneously makes developers aware of changes in their community [12] and provides a way to locate experts [15].

The knowledge shared in these communities may be explicit or tacit. Explicit knowledge can be found in the artifacts of participants, such as blog posts or answers to questions [9]. Tacit knowledge can be supported through informal mentoring, the sharing of experiences, observation, discussion, and trust within a social media platform [17]. Through building an online identity, developers are able to establish expertise and build trust associated with their identity [14]. With knowledge foraging activities, developers are able to build on the experiences of others and create common spaces where the discussion and transfer of knowledge can occur.

Previous work has found that developers use "rich media" such as screencasts and podcasts to support learning and follow trends in their community [1]. From this work, we know that developers use screencasts across social channels to gain technical knowledge. Yet more research is needed to understand how video hosted in social channels supports documentation and the exchange of technical knowledge.

C. YouTube and Screencasts

Screencasts created by developers are frequently hosted on a larger social platform. In order to understand the implications of video within a social context, it is also important to understand the social aspects of the hosting media channel. YouTube is easily one of the most popular online platforms for sharing videos—over 100 hours of video is uploaded every minute [18]—and developers frequently use it.. For these reasons, we chose YouTube as a focus of our study. Although the use of YouTube has not been studied in software engineering research, it has been studied in other areas.

YouTube has a number of social features, which facilitate interaction and communication amongst users [19]. It allows users to create profiles, maintain individual channels, participate in discussions via comments, and search for material [20].

Researchers have examined YouTube as an educational platform [21], [22] and found that students enjoy using it for content delivery—it lets users discover content and learn at their own pace [23]. Other studies have investigated how to create effective educational video content for the classroom [27], and how YouTube is used as a source of informal "Do It Yourself" and "How-to" videos [28]. Researchers have explored how screencasts are made [25], what techniques are used for sharing knowledge [27], and best practices for creating screencasts [29]. These studies suggest that YouTube provides an online participatory culture which serves as a way to share knowledge between experts and amateurs [30]. Users share content to stay connected to their online community and provide updated sources of new information [28].

This paper offers a first study on how screencasting is used as a social channel for sharing knowledge amongst developers, and how screencasts can be used to support software development and documentation tasks.

III. Methodology

Our research methodology consisted of two phases. In the first phase, we conducted an analysis of screencasts hosted on YouTube. Our goal was to understand how developers share software knowledge when they narrate and navigate their source code in a screen capture, as well as understand the nature of the knowledge shared. In the second phase, we interviewed screencast creators to understand their motivations for and experiences with creating screencast content. Our intent was to gain insights that would inform the practices of others and improve support for creating screencast content.

A. Research Questions

We developed and refined four research questions over the course of this study. RQ1 and RQ2 were addressed by analyzing a sample of videos hosted on YouTube. Insights from interviews with screencast creators were used to answer RQ3 and RQ4.

RQ1: What kinds of program knowledge are captured in screencasts? In previous work [1], we learned that some developers rely on screencasts (and podcasts) to learn new technical information, but we did not discover what types of knowledge are best portrayed through this medium.

RQ2: How are screencasts used for documenting code? We wanted to learn about the techniques developers use to explain and present program code through screencasts. We hoped to discover what level of detail they discuss in screencasts, which kinds of programming features are described, and how the authors talk about and present code.

RQ3: What motivates developers to create screencasts? We wanted to investigate developer motivations for creating screencasts and to gain insights into the factors that drive them to distribute their work through video sharing channels.

RQ4: How do developers produce screencasts and which challenges do they face? We wanted to understand more about the "behind the scenes" production process and uncover the challenges developers face when making screencasts. Through the interviews, we were able to identify both common techniques for screencast creation and areas of friction.

B. Phase 1: Video Analysis

1) Screencast Selection Process: To answer the first two research questions, we searched for and analyzed a sample of videos from YouTube. To establish this sample, we defined a list of inclusion criteria: the video should have a description or title containing the term "Code Walkthrough" or "Code Tour"; the video should present the code for a working program or code snippet; the video should contain audio narration and a recording of the narrator's computer screen; and the video should be hosted on YouTube from November 2013 until March 2014. The terms "walkthrough" and "tour" were chosen to focus our search on videos that showed completed code and because the terms imply guidance through a project.

We excluded screencasts that focused on explaining programming fundamentals, as screencasts aimed at teaching the basics of programming have a very different purpose. Rather, we were interested in videos a developer would use to learn about unfamiliar technologies or libraries. A standard YouTube search was used with no filter options on a clean browser. By default, YouTube sorts these results by relevance. To select our sample, we watched the beginning of each video in the search results to see if it met the inclusion criteria. The final sample contained 20 videos and represented over 8 hours of footage. Though this represents a fraction of the possible screencasts hosted on YouTube, our findings were well saturated using this sample.

2) Screencast Sample Characteristics: Table I provides descriptions and links to the selected videos[3]. The topics of these screencasts include Arduino projects, Django, the Corona SDK, Google App Engine, VB.Net, C#, C++, Java and Unity programs. The videos ranged in age from eight years to a few months. Total views for each screencast ranged from a few dozen to several thousands; at the time of our study, eight of the videos had over three thousand views. Table I shows some metrics on the videos we analyzed. In the following, specific screencasts are denoted as V# (as per Table I).

In 14 of the videos, the narrator had authored the program featured in the screencast. Two videos (V17 and V20) involved multiple narrators discussing the code in the video. V17 was aimed at onboarding developers onto an open source project, while V1 and V16 demonstrated examples of how to integrate their servers into a program. V20 was a weekly video podcast that explained how to use the Corona SDK.

3) Screencast Analysis: Screencasts were analyzed using a qualitative coding methodology: we used open coding to systematically review the videos [31]. To develop a preliminary set of codes, two of the paper's authors individually reviewed

[3]Click on the URL links in the PDF to navigate to the videos

Number	Title	URL Links	Length	Views	Comments
V1	Simple Chat Walkthrough with Unity	0d-wY65uVGw	14:51	3184	7
V2	Code Walkthrough - Cafe Townsend Robotlegs for Corona SDK	6MpuB654_hM	59:15	658	12
V3	Minecraft Launcher - Code Walkthrough	UiST0tcOWvU	35:07	6339	93
V4	Django - high level walkthrough	qUpiWWjOfRw	12:21	263	2
V5	Addressbook: A walkthrough of a simple AppEngine application	nwn3YY6cyEQ	11:09	25848	2
V6	Ruby and Rails Code-Walkthrough/Tutorial	djApduemlf4	56:33	3251	3
V7	Asteria Plugin Source Code Tour	jGR0EVYc_Bo	23:43	84	1
V8	Asteria World Generator Tour	mn_wW8uZ6eQ	05:59	50	0
V9	Tour of Mega Happy Sprite source Code	G1DbLOVs7UM	14:44	138	1
V10	A High Level Cruise Through Ruby MRI's Source Code	0npv906IQag	33:20	7419	21
V11	Arduino Code Walkthrough for the Talking Breathalyzer Portable Mode Part 1	wg_gNs3Xxq4	14:00	726	1
V12	Arduino Code Walkthrough for the Talking Breathalyzer Portable Mode Part 2	UdDr9QquiLc	04:39	455	0
V13	Intro to Cocos2d Tutorial Part 2: Code Walkthrough	T1-yARGKhXU	92:53	8616	16
V14	3 Minute Code Walk through of Part 1 of the AR Programming Series	kwY-8mAyixU	03:04	507	1
V15	Arrow M2M kit - node code walkthrough Part 1	WCuQOjD8w_E	06:06	46	0
V16	Blackberry MBO Application Code Walkthrough	T1f4xXoRDG8	07:36	104	0
V17	OpenDaylight OVSDB Developer Getting Started - Code Walkthrough 2 (Java APIs)	3-jCTvNRJS0	26:20	*	0
V18	Flocking Code Walkthrough	OPuYYLEyz-A	15:46	41	0
V19	VB.Net Web Crawler/Spider Source Code Walkthrough	iep-z1KXRN8	08:19	6334	7
V20	Corona Geek #49 - Creating A Simple Game (Code Walkthrough)	O130d8ioFS4	64:15	*	4

TABLE I
THE YOUTUBE VIDEOS ANALYZED. ALL THE YOUTUBE VIDEO URLS ARE CLICKABLE AND FOLLOW THE FORMAT OF "HTTPS://WWW.YOUTUBE.COM/WATCH?V=". VIEW AND COMMENT COUNTS AS OF NOV 2013. *VIEW COUNTS UNAVAILABLE FOR GOOGLE HANGOUTS.

and coded a set of two videos. Then we came together to compare findings and discuss what we had observed.

Through multiple iterations, and after coding a set of six videos together, we developed a common set of codes and documented them in a code book, including definitions and examples of each code [31][4]. Using this code book, we each reviewed the same videos until an inter-rater reliability of 80% was achieved [32]. This threshold was met after two iterations, and then we coded the remainder of the selected videos separately. Throughout this process, we continued to discuss the emerging categories and their relationships, ultimately establishing 21 codes to describe the techniques used by developers in the screencasts. The findings from this process helped us develop our interview questions for the second phase of the study.

C. Phase 2: Interviews with Developer Screencasters

1) Interviewee Selection Process: For the second phase of our study, we conducted semi-structured interviews with screencast creators. We contacted six of the authors of the screencasts we analyzed in the first phase of the study, but only two people responded. The other 14 videos we analyzed were hosted under corporate or inactive accounts. Using the screencast selection criteria from the first phase of the study, we generated a list of additional video authors to contact. In total, 15 more developers were contacted through email, YouTube or social media. 10 developers responded to our requests for interviews and were included in the study.

The interviewees were all English speaking, male, and had formal education in programming or Computer Science. All participants had programming experience outside of academia, though not all were active developers at the time of the study. Three participants were students, five were active software developers, and three of the participants were educators (formerly or currently). Participants were from Europe, the United States, Canada and Australia. While all participants had uploaded at least one video, five of our interviewees had posted over sixty videos.

2) Interview Process: The interviews ranged from 20 to 45 minutes. We used Skype and Google Hangouts for 9 of the 10 interviews, and held 1 interview face to face. We asked participants to tell us how they began making screencasts and describe their production process. Audio was recorded during each interview and later transcribed for analysis. Questions asked included "What prompted you to start posting on YouTube?" and "How do you decide to make a video?"[5]

3) Content Analysis: After transcribing the interviews, two of the paper's authors analyzed each transcript using a content analysis methodology [33]. We met to compare findings and create an initial list of themes. After further analysis, we met again to refine the codes and organize them into themes, along with relevant examples and quotes from the transcripts. Through this method, we identified themes that appeared in multiple interviews [34].

IV. FINDINGS

In this section, we present our findings organized around the four research questions. In the first phase of our study, we analyzed screencasts to answer research questions 1 and 2. In the second phase, we interviewed screencast creators to answer research questions 3 and 4. Our findings explore how developers present software in screencasts and their experiences creating them. Throughout the findings, we draw connections to the relevant literature. In the following, screencasts are denoted as V# (as per Table I) and interview participants are denoted as P#.

[4]A copy of our code book can be found at lmacleod.com/docs/ICPC2015_Codebook.pdf

[5]A copy of our interview questions can be found at lmacleod.com/docs/ICPC2015_Interview_questions.pdf

RQ1: What kinds of program knowledge are captured in screencasts?

Developers create screencasts to convey and share technical information. We found a variety of different goals motivating the screencasts we analyzed. The main goals observed are shown below using bold text. We refer the reader to video examples that demonstrate the goals we identified through our analysis.

Sharing Customization Knowledge: Five of the analyzed screencasts showed how programs could be customized. They explained enough of the program functionality and development details (such as pertinent features and variables) so that the code could be repurposed or customized. For example, the narrator of V11 showed how other developers could change the sound files used by the program being discussed.

Sharing Development Experiences: In eight of the videos, narrators shared insights on their development experiences and the challenges encountered along the way. Their experiences became a part of the narrative tied to the physical code and its evolution. For example, narrators described ways to expand the program $_{V01,V02,V03,V7,V8,V11,V19,V20}$. This included features that the narrator had not yet implemented, or ways of improving performance. In V19, the narrator explained while executing the program:

> "I didn't use multithreading. At first I didn't have time for it, and then I didn't feel like it was quite necessary. I mean, if you want to add that functionality, you can."

Here the narrator acknowledged that he didn't have time to implement a feature. He also recognized that multithreading may improve performance. This provides context to the audience about the program's development history.

Sharing Implementation Approaches: Narrators use screencasts to share unique problem solving solutions with developers that may face similar problems. For example, V18 presents an implementation of an artificial intelligence animation behavior, using JavaScript and HTML. This video serves as both a demonstration and documentation of the narrator's approach. In another example, V03 demonstrates a custom game launcher developed by the narrator for the popular Minecraft game. The narrator was proud of his program and wanted to share his approach with the Minecraft community. He also explained the challenges he faced creating the user interface.

Demonstrating the Application of Design Patterns: Screencasts are used to demonstrate how design patterns are applied as part of the implementation of a solution. In V02, the narrator stated that he made the video to demonstrate how to implement responsive design principles in Lua. He also wanted to share language-specific knowledge through a screencast. The video demonstrates design principles that rely on an understanding of the Lua programming language through a real world example. The video approach allows the narrator to demonstrate the implementation of these design patterns while executing the finished product.

Explaining Data Structures: Other technical screencasts allow developers to explain language-specific data structures. The goal of these videos is to impart technical knowledge about a specific language, as opposed to implemented programs or patterns. This technical knowledge helps the audience utilize and understand the programming language. For example, V10 includes a walkthrough of the Ruby programming language. In this video, the narrators discussed the low-level implementation of Ruby data structures. The video shows the language's internal structure and commonly used components. In order to create these technical screencasts, the narrators need a deep level of understanding about the subject matter.

We observed many different goals in the screencasts we analyzed, including demonstrating a unique approach to a problem and showing the audience how they could customize a program. These walkthroughs forced the narrator to explain their understanding of a program and how they developed its features.

RQ2: How are screencasts used for documenting code?

By answering RQ1, we were able to gain insights into the broad kinds of programming knowledge captured by screencasts. For RQ2, we drill down to investigate **how** developers portray and interact with code in these videos in more detail. In this analysis, we coded the techniques the narrators used in the screencasts, and then grouped the coded techniques into higher-level themes that represent the key approaches used for documenting code. These key approaches (themes) are shown below using bold text, and for each approach, we include the related coded techniques in italics. In some cases, we give more detailed examples of the coded techniques, and frequently reference the videos that contain the observed technique and approach. We summarize the approaches, corresponding coded techniques, and show more details on their screencast occurrences in Table II.

Goal Setting: In nearly all the screencasts in our sample, we observed that narrators started by *verbally defining the video's purpose*, which was followed up with *an explanation of the limitations (i.e., the scope) of the video and its intended audience*. Lethbridge *et al.* noted that developers will avoid reading documentation that is "complex and time-consuming" [7]. Consequently, it is not surprising that a video's purpose and scope was explicitly stated up front, as doing so allows a developer to quickly judge if the resource applies to their current task.

Live Editing to Showcase Code Changes: A key advantage of screencasts over static documentation is that the audience can visually follow changes made to the code.

We observed that *live code changes* were performed in most of the videos. More specifically, we observed developers

Themes	Coded Techniques	Video Occurrences
Goal Setting	Explaining the limitations (i.e., the scope) of the video and its intended audience	V1, V2, V3, V4, V5, V6, V7, V8, V10, V11, V13, V14, V15, V16, V17, V18, V20
	Verbally defining the video's purpose	V1, V2, V3, V4, V5, V6, V7, V8, V10, V11, V13, V15, V16, V17, V18, V19, V20
Live Editing to Showcase Code Changes	Making live code changes	V2, V3, V5, V6, V7, V8, V10, V11, V13, V14, V19, V20
	Changing control flow or variables	V2, V13, V19, V20
	Introducing bugs	V2, V3, V5, V13, V15, V20
Demonstrations to showcase the execution of the program	Executing the program to demonstrate features to the audience	V1, V2, V3, V5, V6, V9, V13, V16, V17, V18, V19, V20
Referencing Different Levels of Detail	High-level code overview	V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20
	Medium-level focus on sub-block of code	V1, V2, V3, V4, V5, V6, V7, V8, V9, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20
	Low-level focus on a single line of code	V1, V2, V3, V4, V5, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20
	Pointing out element identifiers	V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20
	Referencing return types, parameters	V1, V2, V3, V4, V5, V7, V8, V10, V12, V13, V14, V15, V17, V18, V19, V20
	Referencing line numbers	V1, V4, V5
Browsing the Technical Environment	Making use of file explorers	V1,V2, V4, V5, V6, V7, V9, V11, V13, V17
	Explaining the program structure	V1, V2, V4, V6, V7, V10, V13, V14, V16, V17, V18, V20
	Explaining the technical environment (including libraries, servers)	V1, V2, V3, V4, V5, V6, V8, V10, V11, V12, V13, V14, V16, V17, V20
Provisioning of Additional Resources	Webpages	V1, V2, V4, V6, V10, V11, V13, V16, V19, V20
	Diagrams	V16, V17, V18
	Source Code	V1,V2, V4, V6, V9, V10, V13, V18, V19, V20
	Visual Annotations	V1, V3, V6, V10, V11
Mapping Execution and Code to Code	Connection between the demonstration and code	V2, V5, V6, V9, V13, V15, V16, V17, V18, V19, V20
	Linking code segments together	V1, V2, V3, V5, V7, V9, V10, V11, V12, V13, V15, V16, V17, V18, V19, V20

TABLE II
THE CODES AND THEMES (CLUSTERED CODES) CREATED TO ANSWER RQ2: HOW DEVELOPERS USE SCREENCASTS TO DOCUMENT CODE

making *changes to a program's control flow or to specific variables* $_{V2,V13,V19,V20}$. We also saw one developer making live code changes to introduce the audience to examples of increasing difficulty $_{V13}$.

Narrators also *introduced bugs* into the program, either intentionally or unintentionally $_{V2,V3,V5,V13,V15,V20}$. For example, in V05 the narrator created an error and explained to the audience how to read the error output. In V20, an error occurred after the narrator had changed a variable—the narrator used this unexpected situation to turn the session into a live debugging exercise. Live editing also offers the audience a glimpse into the narrator's coding style, or it can be used to manipulate the output of a program.

Demonstrating the Execution of a Program: Screencasts allow the audience to witness "live" executed code. Apart from V04 and V14, all of the screencasts in our sample *executed a program to demonstrate features to the audience*. More specifically, narrators used demonstrations as an opportunity to explain software functionality $_{V01,V02,V13,V18,V19,V20}$. We observed that demonstrations were also used to explain specific use cases and data flows within a program $_{V01,V16,V02,V20}$. Some screencasts with a graphical component made use of demonstrations to explain how a program manipulated user input $_{V05,V06,V13}$.

Traditionally, developers execute software to gain insights into details about the program's behavior [35]. The approach used in screencasts to showcase program execution also allows the audience to see "the properties of a running software system" [36], and through screencasts, narrators show both program output and code, allowing the viewer to create mental mappings between output and code [37].

Referencing Different Levels of Detail: We observed narrators describing features and components using various levels of detail. Features that ran across many files were summarized by the narrators in all the videos using *high-level* descriptions. For example, the narrator of V02 described how the program handled views:

> "When the user successfully logs in, I need to react to it, Cafe Townsend needs to orchestrate something to happen. That is ... show the employee view. It doesn't tell the employee to load anything, it's assumed that view will take care of all it needs to handle."

Narrators provided high-level overviews of program structure and features, but also described code in terms of functional groupings. We coded references to blocks of code, such as a logical sequence that spanned multiple lines, at a *medium level* of detail. Narrators used this level of detail in all the videos (except V10) to quickly summarize sections of code while still providing some technical details. For example, the narrator of V03 briefly described the control flow of four lines in an if statement:

> "First I want to make sure that the directory exists for where we are going to install [the launcher]."

In contrast, when narrators referred to in-line, language-specific information, these were coded as *low-level*. The narrators in all the videos referred to one or more low-level details. Frequently this meant reading the code aloud, verbatim. For example, with respect to the following source code:

```
private Room currentRoom = null;
```

V01 read this aloud as follows:

> "There is also a current room property with the data type of room and it's set to null."

Within these low-level descriptions, narrators would reference specific *line numbers* or mention *element identifiers* and explain their functionality to the audience. For example, the narrator of V04 picked out the variables "Name", "User" and "Password" to explain how to set up a connection to a local database. In addition, they highlighted *return types and parameters* to provide contextual information about control flow and focus the audience's attention on relevant technical implementations.

Browsing the Technical Environment: The environment in which a program is run can be essential to understanding the program itself. We saw narrators in all but three of the videos exploring the *technical environment* to give the audience a spatial understanding of the supporting libraries or tools needed for the program (e.g., launching and connecting to a server).

One of the benefits of screencasting is that the narrator is able to provide context for the development environment, such as how to set up a program, and information about its structure. From our analysis, we saw narrators describe where code fit within the *structure of a program* using various levels of scope $_{V1,V2,V4,V6,V7,V10,V13,V14,V16,V17,V18,V20}$. We also observed the *file explorer* being used for browsing a project and describing file organization $_{V1,V2,V4,V5,V6,V7,V9,V11,V13,V17}$. These explorations provide context to the audience, letting them establish links between code and features of the program [27].

Provisioning of Additional Resources: Screencasts exist as documentation accessible to software developers via the Internet. From our analysis, we found narrators incorporating other forms of documentation, such as *Webpages* for audience members to find additional documentation $_{V1,V2,V4,V6,V10,V11,V13,V16,V19,V20}$, and *diagrams* to provide a visual aid $_{V16,V17,V18}$. *Source code* was also frequently referred to in the screencasts. It was provided to the audience as a working example that they could manipulate on their own $_{V01,V02,V04,V06,V13,V18,V19,V20}$. A notable exception to this was V03: the narrator explained how he specifically did not make his source code available.

To fix errors and share links, narrators used YouTube's *visual annotation* feature $_{V01,V03,V06,V10,V11}$. We also observed two narrators supplementing existing text-based documentation with screencasts $_{V16,V20}$.

Mapping Execution to Code and Code to Code: A benefit of screencasts is that they allow the narrator to walk the audience through code execution and source. We observed narrators *making connections between the execution and the code* to show the audience where program features resided $_{V2,V5,V6,V9,V13,V15,V16,V17,V18,V19,V20}$. By linking together code elements and project structure, narrators built up the audience's understanding between *segments of code* and explained their relationships $_{V1,V2,V3,V5,V7,V9,V10,V11,V12,V13,V15,V16,V17,V18,V19,V20}$.

RQ3: What motivates developers to create screencasts?

In the second phase of our study, we interviewed developers who created and distributed screencasts. From our interviews, we found that developers create screencasts to generate an online identity, to promote themselves, to help them learn, and to give back to the community by sharing knowledge with others. Developers also create screencasts as an alternative to blogging. We use quotes from the interviews to demonstrate these motivational themes, referencing specific interviewees using P#.

To Build an Online Identity: As mentioned in the background literature, developers build an online identity through online participation, which in turn is associated with their skills and projects [14]. We observed that YouTube helps developers create an online identity and build "social capital" [38]. For example, P05 made screencasts to improve his credibility with potential employers:

> "The biggest impact those videos had was on credibility. The fact that they exist was a sign that I'm serious and I should be taken seriously."

Screencasts require time and effort to create. By creating these artifacts, developers codify their knowledge and establish themselves as being knowledgeable in an area.

To Promote Themselves: Once developers have created an identity, they put it to work for personal gain. The literature shows that developers experience positive benefits (e.g., finding jobs) from building their online identity in developer social networks [4]. Developers promote themselves by creating an identity tied to a set of skills. They also benefit by curating and sharing knowledge resources with others [3].

In our study, we also found that developers use YouTube to promote themselves for personal or financial gain $_{P03,P04,P05,P10}$. For example, P10 began posting his videos on Udemy as a way to make money. But through YouTube, he reached a bigger audience than with the closed, paid model of Udemy. When he posted his videos on YouTube, P10 said:

> "It went wild. Like I got tens of thousands of people viewing these videos and sending me personal messages about how amazing the course is and they want more... and I have a lot of blog followers now,

and I have a couple thousand YouTube subscribers from thirty."

P07 also used YouTube as a public channel to reach more viewers. He spoke of how this experiment helped promote his Udemy course and, as a result, benefited him financially.

P06 published a game to an app store and wanted to share his experiences with other developers. He described how, as a sole developer, he cannot "outspend bigger companies, but you can out teach them in a way." He put his screencast on YouTube to create a relationship with his audience that he felt would benefit him financially. P06 said:

> "And teaching people builds trust like I said and you're kind of giving them something for free in a sense where that's a nice thing and then they feel like they know you. It's a much more natural relationship with your audience."

Some developers also described how they used YouTube to promote their companies or products, not just themselves $_{P04,P06,P07,P10}$.

As a Learning Exercise: Our interviews demonstrated how developers create screencasts to better understand a topic $_{P01,P02,P03,P07}$. For example, P01 used screencasts to improve his skills every week:

> "I just open up the Blender specs and see what developers were working on, what was new, what people might have some issues with. I figure out how to do it on my own and then make a tutorial on it."

Using a resource that was openly available online, P01 focused his screencasts on new developments in the Blender community. He identified not only changes to the technology, but taught himself how to adapt to those new changes. He gained insights, which he then shared in his screencasts.

Other interviewees used screencasts as a way to teach themselves new material. They described feeling more confident in their understanding of the material after creating a screencast $_{P03,P07}$. P07 used screencasts as a way of testing himself on the material:

> "I mean in creating it you have to know something well enough to articulate it. So I'll record myself ... and I'll watch it and I'll say 'dude I don't know what I'm talking about', so I'm going to go research again ..."

This experience of learning from sharing through social media is not isolated to video channels. Other researchers have found that knowledge workers who blog about their profession feel that "they have more expertise... than those who do not blog" [39]. Similarly in the educational literature, creating a blog or screencast forces students to use their "critical thinking, producing and publishing skills" to reflect on their experiences within a topic [40].

To Give Back: We found that developers make videos to document what they wished they had known before they started a task $_{P01,P03,P07,P08}$. Research on how developers use Twitter revealed that developers share their experiences as a way to "spare others from having to go through the same discovery process" [41]. Sharing knowledge through social media is done not only to benefit the developer and their identity, but also to distribute knowledge and contribute to a larger community [42].

Three of our interviewees were teachers $_{P03,P05,P08}$ and two of them spoke of how they started creating videos to help students in their classes $_{P03,P08}$. P03 reflected on his development experiences and created content that addressed the problems he encountered while learning:

> "I kind of wanted to package that nicely and deliver it to people who are where I used to be last year, working on my thesis projects, and were like 'if only I had known."'

This desire to help others has been shown in other literature where developers contribute knowledge with no expectation of direct compensation [43] and share information with altruistic motivations [44].

As an Alternative to Blogging: In our interviews, several developers described making screencasts as an alternative approach to blogging (an activity very popular with many developers these days) as they found creating screencasts to be easier than writing $_{P07,P08,P09}$. As video creators, these interviewees felt that they were comfortable with the information they wanted to present.

Interviewees also described having a personal preference for learning from video instead of text $_{P03,P07,P08}$. As P03 stated:

> "If I had the choice to watch a video, I'll definitely watch the video. It's way faster and a better, smoother learning curve."

RQ4: How do developers produce screencasts and which challenges do they face?

Our final research question concerns developer screencast production processes. Through the interviews, we gained the following insights (in bold text) into how developers prepare, record and edit their screencasts.

Preparing the Screencast: Before recording, interviewees spoke of the importance of organizing their thoughts and sources. Nearly all of the interviewees, with the exception of P03, discussed using an outline or script to gather their thoughts. For example, P06 noted:

> "I did write... basically just an outline, like a Markdown outline, with headers and sub-points and then I didn't actually even end up referencing it. It was more just the thing where I wrote it all out, so then I had all my thoughts organized."

The preparation stage consisted of participants identifying key points for their videos and gathering materials for recording. Interviewees described writing or finding existing code for their screencasts, as well as gathering pertinent images $_{P01,P07,P08}$.

The interviewees' descriptions of the planning process fits with other research on the topic, such as using outlines as planning tools [27]. Some research suggests that the planning stage is the most important in the screencast creation process as it forces the creator to focus their ideas and goals [25]. Screencast creators also need to tailor video for their audience through the planning stage [29].

Recording the Screencast: When recording video, our interviewees used a wide variety of screen capture tools, such as Quick Time, Camtasia and Screenflow. Other software was mentioned for manipulating audio, such as Astro Effects.

While recording videos, the interviewees stressed the importance of splitting clips into short segments $_{P03,P04,P05,P06}$. This seemed to be for two reasons: it limited the possibility of mistakes (which would require a retake) $_{P05,P08}$, and it forced the creators to articulate their ideas clearly and quickly $_{P02,P03,P04,P06,P08}$. As mentioned in the findings for Research Question 2, the literature points to the importance of focused, short screencast videos for learning [29], [45]. Since screencasts are not searchable, short screencasts help the audience locate information [29]. Therefore, this practice benefits the screencast creator and the audience.

Interviewees also stressed the importance of good presentation skills while recording, as P10 mentioned:

> "I make a strong effort to have good audio recording equipment, but I also speak really clearly, and I try to communicate my ideas so that they're really easy to listen to and to follow along."

Being clear when recording videos is important for screencasts [25]. But screencast narrators who speak quickly may be perceived as more energetic and engaging by the audience [45].

Post-production: After finishing an initial recording, interviewees took the time to edit their screencasts. Seven of the ten video authors told us they edited their work to improve the quality of their videos $_{P01,P04,P05,P06,P07,P09P10}$. The editing actions ranged from adding effects, to improving and augmenting the audio. P01 applied a lengthy post-production process to his screencasts:

> "I'll come over to the audio and spend two hours just combing through getting rid of clicks."

Of those who edited their work, interviewees reported placing a high value in the production quality of their screencasts $_{P04,P05,\ P07,P10}$. A minority of the participants expressed that they did not do any editing and simply watched their videos for clarity before putting them on YouTube $_{P02,P03,P08}$. As P08 noted:

> "I have full respect for the people who actually go in and edit their video, because that to me is just so painful. I find it quicker just to do a good take."

V. Discussion

Modern software documentation exists on the Web in an ever-changing state. It is hosted on question and answer sites like Stack Overflow [46], in public repositories like GitHub [14], and on Wikis [3]. Our exploratory study shows how YouTube contributes to documentation that is embedded within the social developer ecosystem.

Throughout this paper, we have provided what we believe to be a first investigation of the motivations and techniques used by developers for creating and hosting screencasts on YouTube. Our study showed that developers use video to create a permanent record of their understanding of and experiences with a project. This includes design decisions and the challenges faced during development. Our analysis of a sample of screencasts revealed how developers describe code through video. We observed developers executing programs to explain relationships between the code and the program's output. This allows the audience to draw connections between concepts and processes in the program [27]. Furthermore, we saw that developers also focused the audience's attention on specific variables and elements in the code. The sections of code that the narrator chose can be considered examples of "beacons" [47], which gives the audience a better understanding of the program.

Unlike other documentation methods that are primarily text based, screencasts allow developers to share their coding practices through live interactions. These video captured interactions allow developers to present tacit knowledge would be otherwise difficult to present through textual methods. We observed developers using screencasts to share tacit knowledge by sharing their insights and demonstrating processes, such as setting up the technical environment for a project. They also demonstrated executable features of a program and their personal coding style.

It is important to note the limitations of screencasts. Our interviewees cited the lack of searchability as a barrier to finding relevant information in video. Audience members are faced with the choice of watching the whole video and hope that it contains the information they are looking for, or moving on to a different piece of content. Although YouTube has a mechanism for linking to a specific point in a video, we did not see this used in our analysis. Another limitation of screencasts is that some forms of information are more easily digested as text. Specifically, explicit and formal knowledge is easier to codify in a written form [17].

As in previous work that explored how developers use social channels, our interviewees reinforced that developers shape their online identity by choosing what content to share [4]. This was done for a number of reasons, including altruistic and commercial motivations. We found that developers use YouTube to supplement their activities on other platforms and to share screencasts.

VI. Limitations

A primary limiting factor of this study is the generalizability from our sample of 20 videos and 10 interviews. The potential population space contains thousands of videos of screencasts—just considering YouTube alone. However, our findings were well saturated after 20 screencasts and 10 interviews, and as a first study on this topic, reveal interesting insights and provoke future research into this phenomenon.

We relied on YouTube as the source for the videos in our study. Interviewees spoke of using other video sharing platforms, so the results of this study may be biased towards the type of content posted on YouTube. We selected YouTube because of the social features it offers and its widespread adoption. There are a number of other video sharing methods on the Internet, although some of these are behind paywalls. This research only considered videos that were free of charge and openly available. It is possible that these factors (i.e., "free" content, storage and distribution) may have impacted our findings.

VII. Future Work

There are a number of directions to expand this work in future studies. The following themes are centered on work pertaining to the creation of screencasts for developers, as well as addressing how developers use screencasts to support software engineering tasks.

A. Why and How Do Developers View Screencasts

While we know that developers publish content on YouTube, we do not know how and to what extent developers use these videos to support development tasks. We also do not know how many people watch these screencasts or how they use them. These questions should be explored through future studies.

B. Screencasting Communities

In gathering our screencast sample, we noted the number of comments on each video. We also saw that screencasters interacted with their audience through YouTube's commenting features. Future work should consider this aspect.

The popular streaming platform, Twitch, has recently launched a 'game development' channel where developers stream themselves developing software and answer questions from the audience, in real time[6]. A number of our interviewees also spoke of popular developer-focused screencasting sites that influenced their work. From this, we know that there are communities built around producing and sharing developer knowledge through screencasts. In the future, we wish to explore these emerging communities and the relationships between content producers and audience members.

C. Effectiveness of Screencasts for Developers

A further avenue of future research addresses the effectiveness of code walkthrough videos. Work in education suggests that screencasts can be an effective delivery method for formal education [29]. The work in this paper focuses on screencasts in an informal setting. Therefore, it is not known if developers and audiences may face different challenges in this environment than in a formal learning setting.

[6]http://www.twitch.tv/directory/game/Game%20Development

D. Tool Support

Our interviewees expressed frustration with the tools they use to produce screencasts. It is clear that there is room for improvement in screencast tool support. But it is unclear how the current tools used by these developers hinder or impact the content they produce. While we found that a variety of tools are being used, none of the tools are seen as ideal by the content creators.

VIII. Conclusions

Through this research, we provided an initial exploration into how and why developers create screencasts and host them on YouTube, a social video sharing channel. In our study of 20 videos created by developers, we identified high-level goals and techniques for creating such screencasts. These techniques include demonstrations of code, describing code functionality in different ways, and providing an audience with source code that is integrated with live demonstrations of the execution of that code.

The screencast creators we interviewed described how they use screencasts to learn, document their code, and contribute to their online identity and self-promotion. However, they expressed frustration with their current tool support.

The limitations surrounding our qualitative methodology and sample size means that more work must be done to assess the generalizability of our findings to other screencasts. Overall, this work provides a first exploratory study on how developers currently create video-based documentation for developers. In the future, we hope to explore the effectiveness of these types of videos for educational and knowledge transfer purposes, and to study how such videos contribute to improving software documentation and program comprehension.

Acknowledgments

We thank our interview participants for sharing their insights with us, as well as Cassandra Petrachenko and Alexey Zagalsky for editing support and discussions about our research.

References

[1] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "The r evolution of social media in software engineering," in *Proc. of the on Future of Software Engineering*. ACM, 2014, pp. 100–116.

[2] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider, "Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators," in *Proc. of the 2013 Conference on Computer Supported Collaborative Work*. ACM, 2013, pp. 103–116.

[3] J. Udell. (2004, Nov.) Name that genre: screencast. [Online]. Available: http://jonudell.net/udell/2004-11-17-name-that-genre-screencast.html

[4] R. Brooks, "Towards a theory of the comprehension of computer programs," *Intl. Journal of Man-Machine Studies*, vol. 18, no. 6, pp. 543–554, 1983.

[5] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *Software, IEEE*, vol. 20, no. 6, pp. 35–39, 2003.

[6] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: understanding the decisions of open source contributors," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2010, pp. 127–136.

[7] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow," *Georgia Institute of Technology, Tech. Rep*, 2012.

[8] C. Treude and M.-A. Storey, "Effective communication of software development knowledge through community portals," in *Proc. of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ACM, 2011, pp. 91–101.

[9] T. Räisänen and H. Oinas-Kukkonen, "A system architecture for the 7c knowledge environment," *Information Modelling and Knowledge Bases XIX*, vol. 166, p. 217, 2008.

[10] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *Proc. of the FSE/SDP Workshop on Future of Software Engineering Research*. ACM, 2010, pp. 359–364.

[11] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?: Nier track," in *Software Engineering (ICSE), 2011 33rd Intl Conference on*. IEEE, 2011, pp. 804–807.

[12] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," in *Proc. of the ACM 2012 Conference on CSCW*. ACM, 2012, pp. 1277–1286.

[13] E. Wenger, "Communities of practice and social learning systems," *Organization*, vol. 7, no. 2, pp. 225–246, 2000.

[14] A. Capiluppi, A. Serebrenik, and L. Singer, "Assessing technical candidates on the social web," *Software, IEEE*, vol. 30, no. 1, pp. 45–51, 2013.

[15] S. Panahi, J. Watson, and H. Partridge, "Social media and tacit knowledge sharing: Developing a conceptual model," *World Academy of Science, Engineering and Technology*, no. 64, pp. 1095–1102, 2012.

[16] YouTube. (2013, may) Youtube press statistics. [Online]. Available: http://www.youtube.com/yt/press/statistics.html

[17] N. B. Ellison, "Social network sites: Definition, history, and scholarship," *Journal of Computer-Mediated Communication*, vol. 13, no. 1, pp. 210–230, 2007.

[18] I. Duncan, L. Yarwood-Ross, and C. Haigh, "Youtube as a source of clinical skills education," *Nurse Education Today*, vol. 33, no. 12, pp. 1576–1580, 2013.

[19] C. Snelson, "YouTube across the Disciplines : A Review of the Literature," *Journal of Online Learning and Teaching*, vol. 7, no. 1, pp. 159–169, 2011.

[20] S. A. Azer, "Can youtube help students in learning surface anatomy?" *Surgical and Radiologic Anatomy*, vol. 34, no. 5, pp. 465–468, 2012.

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

[22] W. Sugar, A. Brown, and K. Luterbach, "Examining the anatomy of a screencast: Uncovering common elements and instructional strategies," *The Intl. Review of Research in Open and Distance Learning*, vol. 11, no. 3, pp. 1–20, 2010.

[23] 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*, pp. 1–21, 2010.

[24] S. Mohorovicic, "Creation and use of screencasts in higher education," in *2012 Proc. of the 35th Intl. Convention on Information and Communication Technology. Electronics and Miro-electronics*. IEEE, 2012, pp. 1293–1298.

[25] J. Oud, "Guidelines for effective online instruction using multimedia screencasts," *Reference Services Review*, vol. 37, no. 2, pp. 164–177, 2009.

[26] H.-J. Paek, T. Hove, H. Ju Jeong, and M. Kim, "Peer or expert? the persuasive impact of youtube public service announcement producers," *International Journal of Advertising*, vol. 30, no. 1, pp. 161–188, 2011.

[27] K. Charmaz, *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. Pine Forge Press, 2006.

[28] K. M. MacQueen, E. McLellan, K. Kay, and B. Milstein, "Codebook development for team-based qualitative analysis," *Cultural Anthropology Methods*, vol. 10, no. 2, pp. 31–36, 1998.

[29] K. Krippendorff, *Content Analysis: An Introduction to its Methodology*. Sage, 2012.

[30] J. F. Gubrium, *The Sage handbook of interview research: The complexity of the craft*. Sage, 2012.

[31] B. Cornelissen, A. Zaidman, A. Van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 684–702, 2009.

[32] T. J. Biggerstaff, B. G. Mitbander, and D. Webster, "The concept assignment problem in program understanding," in *Proc. of the 15th Intl. Conference on Software Engineering*. IEEE Computer Society Press, 1993, pp. 482–498.

[33] M. Lee, S. Pradhan, and B. Dalgarno, "The effectiveness of screencasts and cognitive tools as scaffolding for novice object-oriented programmers," *Journal of Information Technology Education: Research*, vol. 7, no. 1, pp. 61–80, 2008.

[34] J. Swart and N. Kinnie, "Sharing knowledge in knowledge-intensive firms," *Human Resource Management Journal*, vol. 13, no. 2, pp. 60–75, 2003.

[35] C. Treude, F. Figueira Filho, B. Cleary, and M.-A. Storey, "Programming in a socially networked world: The evolution of the social programmer," *The Future of Collaborative Software Development*, pp. 1–3, 2012.

[36] L. V. Porter, K. D. Sweetser Trammell, D. Chung, and E. Kim, "Blog power: Examining the effects of practitioner blog use on power in public relations," *Public Relations Review*, vol. 33, no. 1, pp. 92–95, 2007.

[37] W. W. H. Richardson, *Blogs, wikis, podcasts, and other powerful web tools for classrooms*. Corwin Press, 2010.

[38] L. Singer, F. M. Figueira Filho, and M.-A. D. Storey, "Software engineering at the speed of light: How developers stay current using twitter." in *Intl. Conference on Software Engineering*, 2014, pp. 211–221.

[39] L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman, "Knowledge sharing and yahoo answers: Everyone knows something," in *Proceedings of the 17th Intl. Conference on World Wide Web*. ACM, 2008, pp. 665–674.

[40] M. Lindvall and I. Rus, "Knowledge management in software engineering," *IEEE Software*, vol. 19, no. 3, pp. 0026–38, 2002.

[41] M. Levy, "Web 2.0 implications on knowledge management," *Journal of Knowledge Management*, vol. 13, no. 1, pp. 120–134, 2009.

[42] P. J. Guo, J. Kim, and R. Rubin, "How video production affects student engagement: An empirical study of mooc videos," in *Proc. of the First ACM Conference on Learning @ Scale Conference*, ser. L@S '14. New York, NY, USA: ACM, 2014, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/2556325.2566239

[43] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," in *Proc. of the SIGCHI Conference on Human factors in Computing Systems*. ACM, 2011, pp. 2857–2866.

[44] M.-A. Storey, "Theories, Methods and Tools in Program Comprehension: Past, Present and Future," *13th Intl. Workshop on Program Comprehension (IWPC'05)*, pp. 181–191. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1421034