Empir Software Eng (2017) 22:1478–1507
DOI 10.1007/s10664-017-9501-9

Documenting and sharing software knowledge using screencasts

Laura MacLeod[1] · Andreas Bergen[1] · Margaret-Anne Storey[1]

Published online: 30 March 2017

Abstract Screencasts are used to capture a developer's screen while they narrate how a piece of software works or how the software can be extended. They have recently become a popular alternative to traditional text-based documentation. This paper describes our investigation into how developers produce and share developer-focused screencasts. In this study, we identified and analyzed a set of development screencasts from YouTube to explore what kinds of software knowledge are shared in video walkthroughs of code and what techniques are used for sharing software knowledge. We also interviewed YouTube screencast producers to understand their motivations for creating screencasts as well as to discover the challenges they face while producing code-focused videos. Finally, we compared YouTube screencasts to videos hosted on the professional RailsCasts website to better understand the differences and practices of this more curated ecosystem with the YouTube platform. Our three-phase study showed that video is a useful medium for communicating program knowledge between developers and that developers build their online persona and reputation by sharing videos through social channels. These findings led to a number of best practices for future screencast creators.

Keywords Software engineering · Screencasting · Social media

Communicated by: Christian Bird and Rocco Oliveto

✉ Laura MacLeod
lmacleod@uvic.ca

Andreas Bergen
andib@uvic.ca

Margaret-Anne Storey
mstorey@uvic.ca

1 University of Victoria, Victoria, BC, Canada

1 Introduction

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

Describing how a program works often involves the laborious task of writing text, a process shunned by many developers. Today's developers use social channels such as blogs, community portals, GitHub, Twitter and Stack Overflow to author and share documentation (Storey et al. 2014). Sharing knowledge on these social channels benefits the producer of the documentation as it helps build their online reputation and creates a network of like-minded developers (Singer et al. 2013).

This paper describes a study where we explore how screencasts are used for software documentation. Screencasts are "movies of software"(Udell 2004) that capture a developer's screen as they manipulate code for a specific goal such as to demonstrate the use of a tool or to present *how-to information* that can be difficult to describe in text. Their popularity is growing and they are frequently produced as a source of documentation, but little else is known about how screencasts are used or how they can be improved.

In this paper, we investigate how and why developers create and share screencasts on the Internet. In our previous work (MacLeod et al. 2015), we analyzed a selection of development-related YouTube screencasts to determine how and why they are used for documenting and sharing programming knowledge. We also interviewed YouTube screencast creators to understand how they create their videos, revealing the benefits and challenges one may experience when creating video documentation. This paper extends that work by analyzing a set of screencasts from the popular Ruby on Rails screencast site, RailsCasts[1], and compares how development videos are hosted on YouTube with how they are shared on a more formal screencast sharing site.

Our findings reveal that video is an efficient and useful medium for program documentation aimed at other developers. In addition, the social features of the hosting channel, whether it is YouTube or a self-hosted website, play an important role in screencast development and distribution.

The main contributions of this paper are:

1. an understanding of the types of knowledge developers share through screencasts;
2. an understanding of how developers share knowledge through screencasts;
3. technical insights into how developers create screencasts and the challenges they face;
4. insights into the differences and similarities of hosting screencasts on YouTube (a social channel) compared to a self-hosted professional website; and
5. a list of best practices for future screencast creators.

In the remainder of this paper, we review the literature on the role of documentation within social software development and on using YouTube for sharing knowledge in other domains. Then we outline our research methodology and present the findings from an analysis of 20 YouTube screencasts and 10 interviews with developers who have created YouTube screencasts. We also present a comparison of 7 Ruby on Rails screencasts hosted on YouTube with 7 screencasts hosted on the RailsCasts site. We relate our findings to the existing literature and then address the limitations of our study. As this is a rather new research topic, our paper concludes with a number of directions for future research.

[1] See http://railscasts.com

2 Background

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

2.1 Software Documentation

Documentation is a key component of software development that captures many facets of the development process, including requirements, feature implementations, usage scenarios, code examples, and decision insights. Studies have shown that documentation provides programmers with vital information. For example, Brooks discusses how documentation is used to help programmers understand a program (Brooks 1983). Yet there are many challenges in producing software documentation, such as managing large bodies of documentation, incorporating an appropriate level of detail, and keeping documentation up to date (Lethbridge et al. 2003).

Wikis (Dagenais and Robillard 2010), question and answer sites (Parnin et al. 2012), and technical articles (Treude and Storey 2011) provide crowd-based alternatives for contributing knowledge in online communities. For software documentation, these platforms benefit developers in different ways (e.g., Wikis are easy to create and navigate, question and answer sites remain up to date longer, while blogs generate more fanfare (Treude and Storey 2011)).

A drawback of 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 (Räisänen and Oinas-Kukkonen 2008). Screencasts are becoming an increasingly popular way for developers to describe how others can navigate source code. These videos may capture clips of a developer directly interacting with and manipulating source code as well as showing the executed program. Screencasts provide visual and audio cues that go beyond the information portrayed by static text.

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 section, we review existing research on the social nature of software documentation.

2.2 The Social Nature of Software Documentation

Today's developers work in "social ecosystems around content, media, and developers" (Storey et al. 2014). Social dissemination of documentation promotes collaboration with other individuals (Storey et al. 2010), helps people find answers to questions (Treude et al. 2011), and increases developer awareness among communities (Dabbish et al. 2012).

When developers participate in social ecosystems, online identities are formed (Dabbish et al. 2012). Peers use this information to evaluate other developers and to shape their interactions with them (Capiluppi et al. 2013). The transparency of social media not only helps developers establish identities, it also makes them immediately aware of changes in their community (Storey et al. 2010) and provides a way to locate experts (Wenger 2000).

The creation and consumption of social media content has led to the rise of "communities of practice"—communities that form around shared interests and common goals (Wenger 2000). The knowledge shared in these communities may be explicit or tacit. Explicit knowledge is found in participants' artifacts, such as blog posts or answers to questions (Parnin et al. 2012). Tacit knowledge can be supported through informal mentoring, the sharing of experiences, observation, discussion, and trust within a social media platform (Panahi et al. 2012).

Previous work has found that developers use "rich media" such as screencasts and podcasts to support learning and follow community trends (Storey et al. 2014). 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.

2.3 YouTube and Screencasts

Screencasts created by developers are frequently hosted on a larger social platform and YouTube is easily one of the most popular online platforms for sharing video content. It has a number of social features that facilitate interactions and communication among users (Ellison 2007): it allows people to create profiles, maintain individual channels, participate in discussions (via comments), and search for material (Duncan et al. 2013).

The use of YouTube has not received a lot of attention in software engineering research. One recent exception is the work by Ponzanelli et al. (Ponzanelli et al. 2016) where the researchers developed a technique to extract fragments from YouTube code tutorials. However, YouTube has been studied in other domains. Researchers have examined YouTube as an educational platform (Snelson 2011; Azer 2012) and found that students enjoy using it for content delivery—it lets users discover content and learn at their own pace (Duffy 2007). Other studies have investigated how to create effective educational video content for the classroom (Sugar et al. 2010), and how YouTube is used as a source of informal "Do It Yourself" and "How-to" videos(Lankshear and Knobel 2010). Researchers have explored how screencasts are made (Mohorovicic 2012), what techniques are used for sharing knowledge (Sugar et al. 2010), and best practices for creating screencasts (Oud 2009). These studies suggest that YouTube provides an online participatory culture which serves as a way to share knowledge between experts and amateurs (Paek et al. 2011). Users share content to stay connected to their online community and provide updated sources of information (Lankshear and Knobel 2010).

This paper explores how screencasting is used to share knowledge among developers and how screencasts can be used to support software documentation tasks.

3 Methodology and Research Questions

Understanding the role screencasts play in developer documentation is an emerging research topic that has not been well studied. Our research sought to answer the following questions:

RQ1: *What kinds of program knowledge are captured in screencasts on YouTube?* In our previous work (Storey et al. 2014), we learned that some developers rely on screencasts (and podcasts) to learn new technical information, but we lacked insights into the types of programming knowledge captured through YouTube screencasts. This research question helped us further explore the information being shared with screencasts. By analyzing a set of YouTube screencasts, we were able to identify the different types of screencasts developers create and watch.

RQ2: *How are YouTube screencasts used to document code?* Our goal behind this research question and the analysis of a set of YouTube screencasts was to learn how developers explain and present program code through screencasts, as well as how authors talk about and present their own code. This research question explores how knowledge is presented in screencasts and what techniques are used.

RQ3: *What motivates developers to create screencasts on YouTube?* By conducting interviews with screencast creators, we elicited their motivations for creating screencasts, gaining insights into the factors that drive developers to distribute their documentation through a video sharing channel such as YouTube.

RQ4: *How do developers produce screencasts for YouTube and what challenges do they face?* Through the interviews, we were also able to identify the common techniques developers use and the friction they experience when creating video software documentation.

RQ5: *How do screencasts hosted on YouTube compare to screencasts hosted on a professional platform?* Through this question, we wished to compare screencasts hosted on a professional, purpose-built platform (we selected RailsCasts) with similar types of videos hosted on YouTube. We selected and analyzed two sets of Ruby On Rails videos and coded them using the codes that emerged from our answers to RQ1 and RQ2. Our goal was to explore the similarities and differences of screencasts hosted across these two platforms. We also compare some differences across the two platforms.

3.1 Phase 1: YouTube Video Selection and Analysis

To answer RQ1 and RQ2, we selected and analyzed a sample of videos from YouTube. We used the following inclusion criteria to select an appropriate sample of videos for our analysis:

1. The video was available on YouTube from November 2013 until March 2014 (the data collection phase of this part of the study).
2. The video presented a completed code base or code snippet.
3. The video description or title contained a variant of the terms “Code Walkthrough” or “Code Tour”. We selected these terms as we were interested in videos that documented how to use or adapt code without limiting our sample to any one language or technology.
4. The video contained an audio description with a screen capture of the narrator’s computer. The videos we analyzed showed the narrator’s screen and relevant code, and contained audio narration.

We excluded screencasts that focused on programming fundamentals as screencasts aimed at education have a very different purpose. Rather, we analyzed videos a developer would use to learn about unfamiliar technologies or libraries. A standard YouTube search was used with no filter options in a fresh browser session. By default, YouTube sorts these results by relevance, which “includes prioritizing videos that lead to a longer overall viewing session over those that receive more clicks”(YouTube 2016). Further details on the algorithm YouTube uses to rank videos by relevance are not publicly available. To see if videos met our inclusion criteria, we viewed the start of each video returned in the search results. The final sample contained 20 videos and represented over 8 hours of footage. Although this represents a fraction of the possible screencasts hosted on YouTube, our findings were well saturated using this sample.

The top half of Table 1 provides descriptions and links[2] to the selected videos for the first phase of our study. The topics of these 20 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 (at the time of our study). Table 1 shows additional metrics on the videos we analyzed. In the following, specific screencasts are denoted as V# (as per Table 1). To develop a preliminary set of codes, two of our researchers used a qualitative open coding methodology (Adolph et al. 2011) and independently reviewed and coded a set of two videos—we agreed beforehand to code for what we saw or heard in the videos.[3] Then we came together to compare findings and discuss what we had observed. This process occurred over multiple sessions.

Through multiple coding sessions, we established a set of defined codes and documented them in a code book.[4] While developing the book, we discussed the emerging codes, ultimately establishing 21 codes to describe the techniques used by developers in the screencasts. To ensure consistency in applying the codes, both coders reviewed a subset of videos until an inter-rater reliability of 80 % was achieved (MacQueen et al. 1998). After this point, we coded the remaining videos separately.

To answer RQ2, we iteratively grouped the codes to sort them into themes. This process involved creating a Post-it note for each code and arranging them on a board. We then grouped the Post-it notes spatially to visually denote relationships and similarities Fig. 1.

3.2 Phase 2: Interviews with YouTube Developer Screencasters

For the second phase of our study, we conducted semi-structured interviews with 10 screencast creators that hosted their videos on YouTube. We contacted 6 authors of the 20 screencasts we analyzed in the first phase of the study, but only 2 people responded. The other 14 videos we analyzed were hosted under corporate or inactive accounts and the authors could not be reached. To locate additional participants for our interviews, we used similar screencast selection criteria from the first phase of the study and generated a list of 15 additional video authors. In total, we contacted 21 developers through email, YouTube, or social media—10 of these 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 the UK, the United States, Canada, and Australia. While all participants had uploaded at least one video, five of our interviewees had posted over sixty videos.

The interviews ranged from 20 to 45 minutes. Audio was recorded and later transcribed for analysis.[5] We analyzed each transcript using a content analysis methodology (Krippendorff 2012), identifying themes that appeared across multiple interviews (Gubrium 2012). As part of the content analysis method, two of our researchers individually analyzed an initial set of four interviews and identified categories—at this point we were looking

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

[3]More in-depth information about our coding methodology can be found in (MacLeod 2015).

[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 http://lmacleod.com/docs/ICPC2015_Interview_questions

Table 1 This table first lists the YouTube videos (V1-V20) that were analyzed to answer RQ1 and RQ2. It then lists the YouTube videos about the Ruby on Rails language (V21-V27) that were analyzed to answer RQ5. All 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

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 walk-through 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
V21	Ruby on Rails Saving Data: Learn with Codecademy	qKhb3WGd0lM	17:20	1238	6
V22	Ruby on Rails Getting,Started: Learn with Codecademy	XqnEM7V9SjA	7:01	1024	7
V23	Install,Bootstrap into a Ruby on Rails Application	4zMgLzfprqo	16:21	14258	4
V24	CODE GENIUS - OMG Ruby and, Rails Performance by Aaron Patterson	JMGmaRZtgM8	34:42	34852	23
V25	Learn Ruby,with Codecademy: Putting the From in Formatter Section 2/19	CgwZU_c4q3A	11:11	1547	4
V26	Ruby on,Rails Tutorial Part 1 - Getting Started	UQ8_VOGj5H8	11:09	237657	148
V27	Live,Coding: Introduction to Persistence with Ruby on Rails	p5XdoBQNCNU	38:51	18061	32

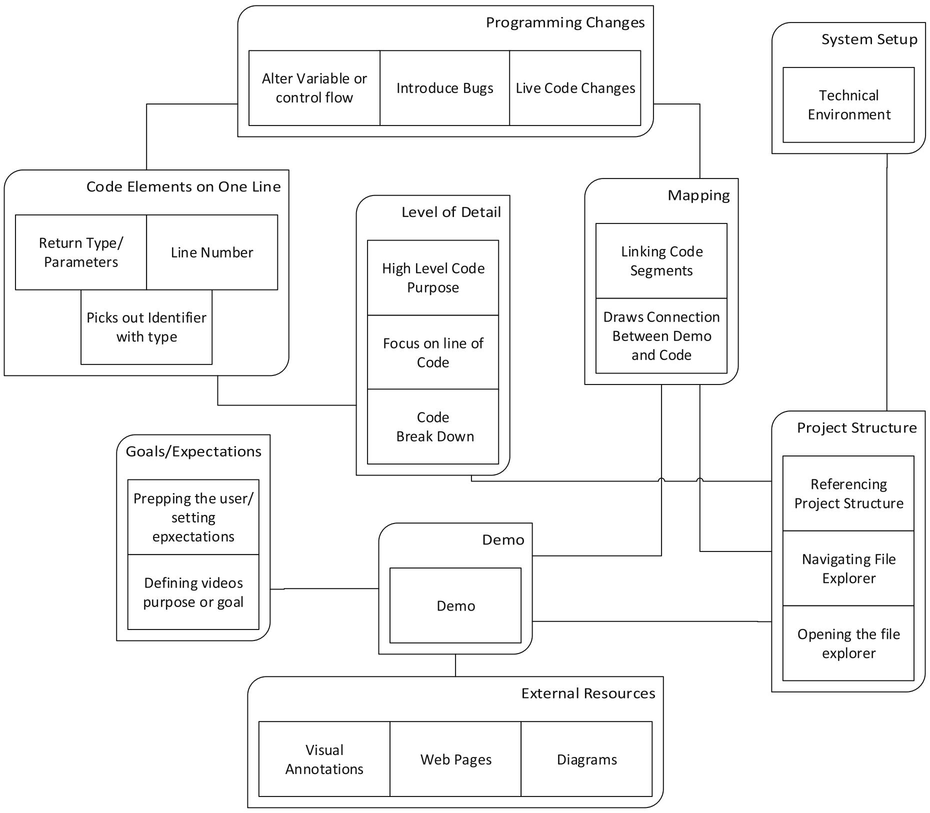


Fig. 1 The resulting map after multiple coding sessions. We used an adjacency coding technique to map different relationships between the codes. As per Charmaz, we tried multiple organizations to distinguish underlying assumptions about each code (Charmaz 2006)

for common ideas and experiences. We then presented our findings to each other, and after several iterations, established a final set of categories.

3.3 Phase 3: Analysis of Ruby on Rails Screencasts Hosted on Different Platforms

To answer our last research question (RQ5), we wanted to compare similar types of screencasts hosted on two different platforms: we selected 7 videos on RailsCasts and 7 additional YouTube videos about the Ruby on Rails programming language. To analyze the screencasts selected in this phase, we used the codes that emerged from our analysis of YouTube videos during Phase 1. We also watched for emergent codes from this new set and added two new codes, as shown in Table 7 and discussed later in the findings.

3.3.1 Ruby on Rails Screencasts Hosted on RailsCasts

RailsCasts is a popular learning resource for Ruby on Rails developers and it is one of three screencast resources linked from the official Ruby on Rails website.[6] According to

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

Table 2 This table lists the RailsCasts videos that were analyzed to answer RQ5. All URLs are clickable and follow the format of "http://railscasts.com/episodes/"

Number	Title	URL Links	Length	Comments	Paid
R231	Routing Walkthrough Part 1	231-routing-walkthrough	13:22	45	No
R232	Routing Walkthrough Part 2	232-routing-walkthrough-part-2	13:26	14	No
R239	ActiveRecord::Relation Walkthrough	239-activerecord-relation-walkthrough	11:45	21	No
R299	Rails Initialization Walkthrough	299-rails-initialization-walkthrough	13:00	9	Yes
R319	Rails Middleware Walkthrough	319-rails-middleware-walkthrough	14:00	7	Yes
R395	Action Controller Walkthrough	395-action-controller-walkthrough	11:00	6	Yes
R397	Action View Walkthrough	397-action-view-walkthrough	15:00	13	Yes

Alexa web tracking, RailsCasts is one of the top 32,000 visited websites on the Internet.[7] While this influenced our decision, we ultimately chose the RailsCasts platform because it contained videos that closely mirrored the inclusion criteria we specified for the YouTube videos in the first phase of our study (i.e., the videos contained similar keywords in their titles and descriptions).

The ability to access content was another reason for choosing RailsCasts. When asked about effective screencasting sites, our interviewees mentioned RailsCasts and Egghead.io, but Egghead.io requires a $200 annual fee to access content behind their paywall, while RailsCasts charges $9 monthly.

RailsCasts hosts over 400 screencasts (published periodically from 2007 until 2013), and while it offers some videos for free, many are behind a paywall (four of the seven videos we analyzed were behind the paywall). In order to access the paid videos, we acquired a paid subscription. For our comparison, we selected the set of videos featured in the site's "Code Walkthroughs" category as this closely mirrored our YouTube search criteria keywords. The videos we analyzed were created by RailsCasts' owner, Ryan Bates. Table 2 shows the characteristics of the videos we analyzed.

3.3.2 Ruby on Rails Screencasts Hosted on YouTube

Rather than just compare the YouTube screencasts from Phase 1 to the RailsCasts screencasts, we decided to use other YouTube screencasts focused on Ruby on Rails. To find additional videos for this comparison, we used the same search criteria from Phase 1 but included the search term 'Ruby on Rails'. We selected an additional seven YouTube videos—three of the videos were created by companies that focus on teaching programming rather than just code walkthroughs.

4 Findings

In this section, we present our findings organized around the five research questions. Our findings explore how developers present software in screencasts and their experiences creating them. Throughout our presentation of the findings, we draw connections to the relevant literature.

[7] http://www.alexa.com/siteinfo/railscasts.com. Alexa rankings as of October 2015.

RQ1: What kinds of program knowledge are captured in screencasts on YouTube? Developers create screencasts and host them on YouTube to convey and share technical information. We found a variety of different goals motivating the screencasts we analyzed. The main goals we observed from the initial 20 YouTube videos we analyzed in Phase 1 are shown using **bold text** and discussed below. We refer the reader to video examples (screencasts are denoted as V# for the YouTube videos (see Table 1) that demonstrate the goals we identified through our analysis. These goals and the videos where they occurred are summarized in Table 3.

Demonstrating the Application of Design Patterns YouTube screencasts are sometimes used to demonstrate how repeatable design patterns are included as part of the implementation of a solution. In the V02 screencast, the narrator stated that they made the video to demonstrate how to implement responsive design principles in the programming language Lua. They 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 hosted on YouTube allow narrators 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 understanding of the subject matter.

Table 3 The types of technical information shared by the YouTube videos we analyzed in Phase 1 to answer RQ1, as well as the YouTube and RailsCasts videos we analyzed in Phase 3 to answer RQ5

Types of Program Knowledge Captured in Screencasts:	YouTube RQ1	YouTube RQ5	RailsCasts RQ5
Sharing Customization Knowledge	V1, V5, V11, V12, V15, V16	V21, V22, V23, V24, V25, V26, V27	R319, R395, R397
Sharing Development Experiences	V1, V2, V3, V7, V8, V11, V19, V20	V21, V22, V23, V24, V25, V26, V27	
Sharing Implementation Approaches	V3, V4, V6, V9, V13, V14, V17, V18	V21, V22, V23, V24, V26, V27	R231, R232, R299, R319, R395, R397
Demonstrating the Application of Design Patterns	V2, V5	V21, V22, V24, V25, V26, V27	
Explaining Data Structures	V4, V10	V23, V24	R239

Sharing Implementation Approaches Narrators use screencasts on YouTube to share unique problem-solving solutions with developers that may face similar issues. 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. By sharing implementation approaches, the narrator discussed how they developed a solution to a particular problem. In another example, V03 demonstrates a custom game launcher developed by the narrator for the popular Minecraft game. The narrator was proud of their program and wanted to share the approach with the Minecraft community. They also explained the challenges they faced creating the user interface.

Sharing Development Experiences In eight of the YouTube videos analyzed in Phase 1, narrators shared insights on their development experiences and the challenges encountered along the way. Their experiences became part of the narrative tied to the physical code and its evolution. The narrators described ways to expand the program. This included features that the narrator had not yet implemented or ways of improving performance. For example, V19's narrator explained the following while executing the program in the screencast: *"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."* Here the narrator acknowledged they didn't have time to implement a feature. They also recognized that multithreading may improve performance. This provides context to the audience about the program's development history.

Sharing Customization Knowledge Five of the YouTube screencasts from Phase 1 show how programs can be customized. The narrators explained enough of the program functionality and development details (such as pertinent features and variables) so that one could repurpose or customize the code. For example, V11's narrator showed how developers could change the sound files used by the program they discussed in the screencast.

In summary, we observed many different goals in the YouTube screencasts analyzed in Phase 1, 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 YouTube screencasts used to document code? For RQ2, we investigated how narrators portray code documentation through video. For this analysis, we coded the techniques the narrators used in the screencasts, and then grouped the coded techniques into high-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*. We summarize the approaches, corresponding coded techniques, and show more details on their screencast occurrences in the 20 YouTube videos from Phase 1 in Table 4.

Goal Setting In nearly all the YouTube screencasts in our sample, we observed that narrators started by *verbally defining the video's purpose*, which was followed by *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" (Lethbridge et al. 2003). Consequently, it is not surprising that a video's purpose was explicitly stated first. This allows a viewer to quickly judge if a video 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

Table 4 This table summarizes how the narrators of the YouTube videos used screencasts for documenting their code (RQ2)

Theme	Coded Techniques	YouTube Videos
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 to Code and Linking 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

observed that *live code changes* were performed in most of the YouTube videos in our sample. More specifically, we observed developers making *changes to a program's control flow or to specific variables* in four videos. We also saw V13's narrator making live code changes to introduce the audience to examples of increasing difficulty.

Narrators also *introduced bugs* into the program in six of the videos, either intentionally or unintentionally. For example, V05's narrator created an error and explained to the audience how to read the error output. In V20, an error occurred after the narrator renamed a variable and they 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 witness changes to the output of a program as code changes are made.

Demonstrating the Execution of a Program Screencasts allow the audience to witness "live" executed code. The majority 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 in six videos. We observed that demonstrations were also used to explain specific use cases and data flows within a program in 4 of the 20 videos. In these situations, narrators were mapping references to the code, the executable, and external resources to share information. By connecting these different sources, they shared their understanding of the program. The approach used in screencasts to showcase program execution also allows the audience to see "the properties of a running software system" (Biggerstaff et al. 1993), and through screencasts, narrators show both program output and code, allowing the viewer to create mental mappings between output and code (Lee et al. 2008).

Referencing Different Levels of Detail We observed narrators describing features and components using various levels of detail. Features that were distributed across many files were summarized by the narrators in all but one of the videos using *high-level* descriptions. For example, V02's narrator 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 abstraction. Narrators used this level of detail in 19 of the videos to quickly summarize sections of code while still providing some technical details. For example, V03's narrator 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 these 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;                (1)
```

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, V04's narrator 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 executes can be essential to understanding the program itself. We saw narrators in all but three of the YouTube 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. In 12 of the videos we analyzed, narrators described where code fit within the *structure of a program* using various levels of scope. In 10 of the 20 videos, we observed the *file explorer* being used for browsing a project and describing file organization. These explorations provide context to the audience, allowing them to establish links between code and program features (Sugar et al. 2010).

Provisioning of Additional Resources Screencasts exist as documentation accessible to software developers via the Internet. In the YouTube videos we analyzed, narrators incorporated other forms of documentation, such as *webpages*, for audience members to find additional documentation in 10 videos, and *diagrams* to provide a visual aid in 3 videos.

Source code was also referred to in 8 of the 20 screencasts. It was provided to the audience as a working example that they could manipulate on their own. A notable exception to this was V03: the narrator explained how they specifically did not make their source code available.

To fix errors and share links, the narrators of 5 videos used YouTube's *visual annotation* feature. We also observed two narrators (of V16 and V20) supplement existing text-based documentation with screencasts.

Mapping Execution to Code and Linking Code to Code A benefit of screencasts is that they allow narrators to walk the audience through source code and execution. We observed 11 narrators *make connections between the execution and the code* to show the audience where program features resided. By linking together code elements and project structure, 16 narrators also built up the audience's understanding of multiple *segments of code* and explained their relationships.

Linking segments together allows developers to connect concepts within the program. Through mapping, developers move between low- and high-level explanations. The work of Von Mayrhauser and Vans refers to this cross-referencing as "relating different abstraction levels...by mapping program parts to functional descriptions" (Von Mayrhauser and Vans 1995). By linking beacons (Brooks 1983), narrators navigate the program's code.

RQ3: What motivates developers to create screencasts on YouTube? In the second phase of our study, we interviewed developers who created and distributed screencasts through YouTube. 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#. We summarize the motivations and indicate which interviewees shared these themes with us in Table 5.

Table 5 This table summarizes the interviewees' motivations for creating screencasts

Why do Developers Create Code Screencasts?	Interviewees
To Build an Online Identity	P05, P06, P07, P08
To Promote Themselves	P03, P04, P05, P07, P10
As a Learning Exercise	P02, P03, P07
To Give Back	P01, P03, P07, P08
As an Alternative to Blogging	P01, P07, P08, P09

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 (Dabbish et al. 2012). We observed that YouTube helps developers create an online identity and build "social capital" (Swart and Kinnie 2003). For example, P05 made screencasts to improve their 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/Others 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 (Singer et al. 2013). Developers promote themselves by creating an identity tied to a set of skills. They also benefit by curating and sharing knowledge resources with others (Treude et al. 2012). Four interviewees also described how they used YouTube to promote their companies or products, not just themselves.

In our study, we also found that interviewees used YouTube to promote themselves for personal or financial gain. Four interviewees mentioned this theme. For example, P10 began posting their videos on Udemy as a way to make money. But through YouTube, they reached a bigger audience than with the closed, paid model of Udemy. P10 spoke of their experience when they posted their videos on YouTube: *"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."*

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

As a Learning Exercise Four of our interviewees discussed how developers create screencasts to better understand a topic. For example, P01 used screencasts to improve their 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 their screencasts on new developments in the Blender community. They identified not only changes to the technology, but taught themselves how to adapt to those new changes. They gained insights, which they then shared in their screencasts.

Other interviewees used screencasts as a way to teach themselves new material. For example, two interviewees described feeling more confident in their understanding of the material after creating a screencast. P07 used screencasts as a way of testing themselves 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" (Porter et al. 2007). 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 (Richardson 2010).

To Give Back We found that developers make videos to document what they wished they had known before they started a task, which four interviewees mentioned. 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" (Singer et al. 2014). 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 (Adamic et al. 2008).

Three of our interviewees were teachers and two of them spoke of how they started creating videos to help students in their classes. P03 reflected on their development experiences and created content that addressed the problems they 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 (Lindvall and Rus 2002) and share information with altruistic motivations (Levy 2009).

As an Alternative to Blogging Several of our interviewees said they produced screencasts instead of blogging, a popular activity with many developers these days—three interviewees mentioned that they found creating screencasts easier than writing. Three interviewees also reported having a personal preference for learning from video instead of text, 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 for YouTube and what challenges do they face? This 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. These techniques are summarized in Table 6.

Preparing the Screencast Interviewees spoke of the importance of organizing their thoughts and sources before recording a screencast. Nearly all 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*

Table 6 This table summarizes the intervieweeh techniques for producing screencasts

How Do Developers Produce Code Screencasts?	Subtopics	Interviewees
Preparing the Screencast		
	Used an Outline	P01, P02 ,P04, P05 ,P06, P07, P08, P09, P10
Recording the Screencast		
	Importance of Breaking Clips into Short Segments	P03, P04, P05, P06
	Debugged live	P01, P08, P10
Post-production		
	Edited Their Screencasts	P01, P04, P05, P06, P07, P09, P10
	Did Not Edit Their Screencasts	P02, P03, P08
	Stressed the Importance of Production Quality	P04, P05, P07, P10
	Frustration with Software	P01, P04, P06, P07, P08, P09

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 the creators identifying key points for their videos and gathering materials for recording. Three interviewees described writing or finding existing code for their screencasts, as well as gathering pertinent images.

The interviewees' descriptions of the planning process fit with other research on the topic, such as using outlines as planning tools (Sugar et al. 2010). 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 (Mohorovicic 2012). Screencast creators also need to tailor videos for their audience through the planning stage (Oud 2009).

Recording the Screencast When recording videos, 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.

Four interviewees stressed the importance of splitting clips into short segments when recording. This seemed to be for two reasons: it limited the possibility of mistakes (which would require a retake), and it forced the creators to articulate their ideas clearly and quickly. The literature points to the importance of focused, short screencast videos for learning (Guo et al. 2014; Oud 2009). Since screencasts are not searchable, short screencasts help the audience locate information (Oud 2009). Therefore, this practice benefits both 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."*

Speaking clearly when recording is important for screencasts (Mohorovicic 2012), but screencast narrators who speak quickly may be perceived as more energetic and engaging by the audience, which can increase audience enjoyment (Guo et al. 2014).

Post-production After finishing an initial recording, interviewees took time to edit their screencasts: 7 of the 10 video authors told us they edited their work to improve the quality of their videos. The editing actions ranged from adding effects, to improving and augmenting the audio. P01 applied a lengthy post-production process to their 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, four interviewees reported placing a high value in the production quality of their screencasts. P10 shared a story of how a man with a hearing disability was able to understand his screencasts because of the audio quality: *"It was really cool to know that my courses were also accessible to disabled people simply because of the effort I put into the quality."*

A minority of the participants (only three) said they did not do any editing and simply watched their videos for clarity before putting them on YouTube, 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."*

A common theme was the participants' frustration with the tools they used to create videos—this was mentioned by six participants. Sources of frustration included poor editing and recording software, slow upload times, poor video quality, as well as adjusting their videos to different screen resolutions. Three interviewees reported that they spent the most amount of time in post-production improving the quality of their work. Though many different tools were used in the sample, none of them were found to be ideal by our participants. The majority of people we interviewed reported doing some form of editing as a quality measure.

RQ5: How do screencasts hosted on YouTube compare to screencasts hosted on a professional platform? To answer RQ5, we set out to compare the key characteristics of YouTube screencasts with more professionally made screencasts on a similar topic hosted on a dedicated website. To do this, we selected a set of seven videos from the popular RailsCasts site (we use R# to refer to specific RailsCasts videos below, see Table 2) and compared them to an additional seven screencasts about Ruby on Rails hosted on YouTube (Table 1 lists the additional videos for this phase).

As mentioned previously, to code this set of Ruby on Rails videos, we applied the codes that emerged from our Phase 1 analysis of YouTube videos. We were interested in learning if the types of knowledge shared were similar across the two platforms (that is, to revisit RQ1), as well if the approaches used to document Ruby on Rails code were similar (that is, to revisit RQ2). We also wished to determine if there were other differences or similarities we could see in terms of the platforms and the features they offered.

Comparing the two platforms for hosting Ruby on Rails screencasts All of the RailsCasts videos started with a short overview of how to download the respective source code from the command line via Git, as well as a description of what was going to be covered in the screencast. As mentioned in Section 3, the RailsCasts videos we analyzed were created and narrated by the RailsCasts owner, Ryan Bates.

The RailsCasts site was specifically designed to showcase developer-focused screencasts. **Every screencast is accompanied by source code, an "ASCIIcast" (a RailCasts video transcribed into text, similar to a blog), and code examples**. Showing examples, such as commands for the terminal, makes it easier for the audience to copy items into their local environment. Compared to YouTube, the majority of RailsCasts videos can only be accessed with a **paid subscription**. However, RailsCasts uses YouTube to share free limited previews of paid content. RailsCasts also allows users to easily **download episodes**. While this can be done with YouTube, it requires extra plugins or steps. This makes the RailsCasts videos more accessible off-line.

The RailsCasts videos we analyzed had links to source code, whereas only a few of the Ruby YouTube videos referenced source code and mentioned how to download it. The RailsCasts narrator frequently mentioned previous screencasts they had created, and in such cases, there were easy-to-use **links to other episodes** at the bottom of each screencast. To capture this new behaviour in our open coding, we added the code "Provisioning of Additional Resources: Screencasts", as shown in Table 4.

Finally, as shown in Tables 1 and 2, the RailsCasts videos were all less than 15 minutes, whereas two of the YouTube videos were longer than 30 minutes. This is not a feature of the platforms per se, but perhaps there is an expectation that videos on Railscasts are relatively short.

How do RailsCasts and YouTube compare for documenting types of knowledge about Ruby on Rails Here we revisit RQ1 to compare the types of knowledge shared on the two platforms. We do see some differences, at least in this small sample. As mentioned, overall, the RailsCasts videos were shorter and more focused on a single type of knowledge (see Table 3).

Several of the YouTube Ruby on Rails screencasts we analyzed in this phase *shared development experiences*, such as covering general "getting started" topics (like V23, V24, V26), or were more focused on "data saving" (V21) or "performance" (V24). In comparison, the RailsCasts videos were not focused on shared development experiences and instead tended to cover *implementation approaches* or *sharing customization knowledge*.

In one YouTube video that was recorded live (V27), the narrator used the end of the screencast to answer questions from audience members. Since this behaviour did not emerge in our analysis of any of the videos in Phase 1, we created a new code, "interacting with the audience", for this behaviour. This was not seen in any of the RailsCasts videos or any other YouTube videos analyzed in the third phase.

How do RailsCasts and YouTube compare in terms of how developers share knowledge about Ruby on Rails through screencasts In terms of **how developers use screencasts to document code (RQ2)**, Table 7 shows that there are many similarities but also some differences between the RailsCasts and YouTube videos.

The main differences were in how information was provided to the audience and how people were directed to additional information. First, we saw very **little execution of the source code** in the RailsCasts videos. Since the narrator was walking through the Ruby on Rails source code, the focus was on the implementation of classes and libraries.

Secondly, while the sole RailsCasts narrator live coded to show how to use Ruby on Rails code, they did not introduce any bugs into the code nor did they change the control flow of the program. However, this could be the style of this one narrator rather than a difference between the two platforms.

Thirdly, compared to the YouTube set, the **RailsCasts videos were visually minimalistic**. The most number of windows ever displayed to the audience was two: an IDE file explorer and the source code window. In the YouTube Ruby on Rails screencasts, the source code was sometimes hard to see due to the number of windows open or the small text size. In the RailsCasts videos, there were no instances of diagrams or visual annotations. The narrator focused on the source code, terminal, and less frequently, on a file explorer window or webpage—any source code and webpages were linked under each RailsCasts video.

Table 7 This table shows the comparison of YouTube Videos and RailsCasts videos that were analyzed to compare how developers use screencasts to document Ruby on Rails code (revisiting RQ2)

Theme	Coded Techniques	YouTube Videos	RailsCasts Videos
Goal Setting	Explaining the limitations (i.e., the scope) of the video and its intended audience	V24, V27	R231, R395, R397
	Verbally defining the video's purpose	V22, V23, V24, V25, V26	R231, R232, R239, R299, R395, R397
Live Editing to Showcase Code Changes	Making live code changes	V22, V23, V24, V25, V26, V27	R231, R232, R239, R299, R319, R395, R397
	Changing control flow or variables		
	Introducing bugs		
Demonstrations to Showcase the Execution of the Program	Executing the program to demonstrate features to the audience	V22, V23, V24, V25, V26, V27	R395, R397
Referencing Different Levels of Detail	High-level code overview	V22, V23, V24 ,V25, V26 ,V27	R231, R232, R239, R299, R319, R395, R397
	Medium-level focus on sub-block of code		R231, R232, R239, R299, R395, R397
	Low-level focus on a single line of code		R231, R232, R239, R299, R319, R395, R397
	Pointing out element identifiers	V22, V26, V27	R231, R232, R239, R299, R319, R395, R397
	Referencing return types, parameters	V23	R231, R232, R239, R299, R395, R397
	Referencing line numbers	V23	R231, R395

Table 7 (continued)

Theme	Coded Techniques	YouTube Videos	RailsCasts Videos
Browsing the Technical Environment	Making use of file explorers	V22, V23, V26, V27	R231, R239, R299, R395, R397
	Explaining the program structure	V22, V27	R231, R232, R239, R395, R397
	Explaining the technical environment (including libraries, servers)	V26	R231, R395, R397
Provisioning of Additional Resources	Webpages	V22, V23, V25, V27	R219, R239, R299, R319
	Diagrams	V24	
	Screencasts		R219, R232, R239, R319, R395, R397
	Source Code		R219, R231, R239, R319, R395, R397
	Visual Annotations		
Mapping Execution to Code and Linking Code to Code	Connection between the demonstration and code	V25, V26, V27	R310, R305
	Linking code segments together		R231, R232, R239, R299, R395, R397
Interacting with the Audience	Interacting with the Audience	V27	

5 Discussion

In this section of the paper, we discuss our results to date, noting that our findings are preliminary as this is an exploratory study in an emerging field. Nevertheless, we are able to suggest some best practices gleaned from the interviews with screencast creators and from our examination of the videos hosted on YouTube and RailsCasts. This is followed by a brief discussion of how screencasts play a role in the socio-technical ecosystems that underlie today's software projects. Although our discussion around the impact of screencasts on a community of practice is speculative, we share these initial insights as we believe this points to some interesting future work. Finally, we discuss the limitations of using screencasts as documentation that emerged from our study.

5.1 Screencast Best Practices

Based on our study, we propose a set of best practices for developers wishing to create screencasts. We draw from our interviews with screencast creators (interviewees were asked what they thought were good and bad characteristics of screencasting), the literature, a cursory analysis of the comments from watchers (as many provided feedback to the videos creators), and our analysis of the screencasts from YouTube and RailsCasts.

1) Design screencasts that are short and sweet In the set of YouTube videos we analyzed, some screencasts were over an hour long, whereas all of the RailsCasts videos we studied were at or under 15 minutes in length. Four interviewees stressed the importance of breaking screencasts into shorter segments in the interest of time. Our interviewees felt that short clips are important because they make editing easier and that sitting through an hour-long clip can be overwhelming for the audience, as P02 mentioned to us: *"I'm sure I've been guilty of it. But that's definitely something that when I'm watching a video, if I got the point in the first few seconds, I really don't want to have another three minutes of the guy kind of explaining the same thing over and over again."* Recent related work by Ponzanelli et al. found through a survey that watchers of screencasts "would either try to scroll it to seek the relevant information (47 %), or give up to find alternative sources (53 %)" when a screencast was too long (Ponzanelli et al. 2016) further justifies the need for shorter screencasts.

2) Stay on topic Five interviewees discussed the importance of not rambling and staying on topic. The educational literature suggests minimizing memory load when creating screencasts (Oud 2009) and not overwhelming the audience with too much information. In the RailsCasts sample, we saw the author frequently link to other screencasts. These **point audience members to additional resources** they can use, but also limit the scope of the current screencast.

3) Show by doing As discussed in the findings to RQ2, nearly all of the videos in the YouTube screencast sample executed the program under discussion. While the RailsCasts sample did not execute the code, the narrator did show how to write source code using the classes or libraries under analysis.

Audience members need to make connections about the expected purpose of the source code featured in a screencast. According to Brooks, executing code allows the audience to create mappings between the code and the problem it solves (Brooks 1983). Previous work also suggests that it provides context and sets audience expectations as to what the program does (Oud 2009). Though code that does not produce easily recognizable output can still be demonstrated through **live coding** for context.

4) Strive for high-quality screencasts All of the videos in both screencast samples contained narration. Interviewees stressed the importance of being able to **understand the person speaking** in the screencast.

As previously mentioned, the RailsCasts sample was visually sparse compared to the YouTube sample. Screencast creators can help their audiences by **minimizing distractions**, both audio and visual. Surprisingly, because of text size or a lack of clarity, it was difficult to read the source code on the screen with a number of screencasts in our YouTube sample. Two of the watchers of the YouTube content even requested higher definition versions of the screencasts (V5, V14). **Readable source code** is important for these kinds of videos.

Other quality issues relate to more than the videos. For example, authors should ensure that included **resource links are robust**, as one of the watchers of V11 mentioned:

> “Hey man, good job on the tutorial, it really helped. Your link to your website is not working FYI. Is there a link I can go to and read over the code or a source that would help me learn more about it? Any feedback would be appreciated. Thanks.”

Videos should also be **well paced and well structured** and authors should not jump around too much, as a watcher of R397 mentioned:

> “I love your videos, however, this video and even the controller walkthrough were extremely fast and jumped around too quickly. I just couldn’t focus on what you were saying. It seemed like I was just trying to figure out what directory you were in the entire time and not paying attention to the code.”

Executing the code can be helpful, but at the same time, if the audience has to wait for something to compile or run, they may lose concentration, as a watcher of V19 mentioned:

> “Thank you, it worked well, besides the waiting part.”

5) Plan ahead As discussed previously, interviewees described using a planning mechanism to organize their ideas. Planning helps the screencast creator focus their goals and allows them to reflect on the audience’s needs. Research suggests that this is the most important stage when producing a screencast (Mohorovicic 2012). Planning also helps tailor content for the audience’s “level of knowledge” (Oud 2009). Indeed, one of the watchers of R397 complained about this very thing:

> “I am not sure which purpose the walkthroughs serve though. They are very generic, moreover experienced rails devs can read the code and figure this stuff out. Novice developers will feel lost anyway.”

Previous work suggests that noise—or confusing information—be filtered out at this stage (Oud 2009). In the planning stage, screencast creators should focus on defining a logical path to follow through the source code. This will help the audience with their mental map of the program and execution flow as the screencast navigates through the project’s structure. This step also allows the screencast creator to plan what resources to provide the audience, such as source code and webpages, and where to include links to these materials.

6) Provide the source code As mentioned in the previous step, screencast creators should think about how they will provide source code to the audience. All of the RailsCasts screencasts contained additional source code to complement the video. Four of the YouTube screencast creators we interviewed also stressed the importance of providing source code to the audience. Many YouTube screencasts included links to source code through the YouTube description. From the literature, it is known that "learning enough to apply new knowledge usually requires active engagement or practice in a realistic context" (Oud 2009). The source code provides a real example which the audience can use to experiment with and build upon.

Summary These best practices address common concerns that were brought up in the interviews with screencast creators and from our analysis of YouTube and RailsCasts videos. They also address a number of challenges that are unique to the developer screencasting experience. A main challenge for screencast viewers is understanding the source code presented on screen. By providing the audience with a clear, well-planned video, it is hoped that these guidelines provide support to future screencast creators.

5.2 Screencasting in a Community of Practice

Modern software documentation exists on the Web in an ever-changing state. It is hosted on question and answer sites like Stack Overflow (Mamykina et al. 2011), in public repositories like GitHub (Dabbish et al. 2012), and on Wikis (Treude et al. 2012). Our exploratory study shows how screencasts hosted on social sites contribute to documentation embedded within the social developer ecosystem. Unlike other documentation methods, many of which are based on text, screencasts allow developers to share their coding practices through live captured interactions. Furthermore, screencasts are not an "island"—they are a cog in either a very structured or loosely connected web of documentation and community authored resources.

The screencasts we analyzed were situated in socially enabled websites: YouTube and RailsCasts both have features that support comments from watchers. We performed a cursory analysis of the comments associated with the YouTube and RailsCasts videos in our sample. This analysis gave us some initial insights into the role comments play in the community. Many of the watchers gave **kudos** to the authors of the videos. For example, one of the watchers of R231 shared how important these videos are:

> "Great episode (as always, actually). I would be highly interested in more episodes like this. I find it really interesting how the internals of Rails work but going through the code on your own can be quite overwhelming sometimes."

Indeed, watching these videos can sometimes even **promote others to create screencasts**, as another watcher of R231 commented:

> "Plus 1 for Mato's suggestion of ActiveRecord internals, especially the differences between Rails 2.3 and Rails 3. Come to think [of it], I should probably get my hands dirty and do it myself."

But more than that, the screencast became a place for developers to **network** with others in the community and **ask for help**. Both the screencast authors and watchers asked for help by posting questions. For example, a watcher of R232 made a plea:

> "Any tips for solving problems like this? [provides a link to Stack Overflow question]"

As with previous research that explored how developers use social media, our interviewees reinforced that developers shape their online identity by choosing what content to share (Singer et al. 2013). This was done for a number of reasons, including **altruistic and**

commercial motivations. And although they don't get paid, developers are happy to go out of their way to help others. However, they do **appreciate credit**, as the author of V3 shared:

> "Sorry it has taken me MUCH longer to do this, but I have posted a link to the code in the description. Feel free to use this code however you want. I'm not making any profit off of it and I feel that anyone should be able to use it as they see fit. If you do copy my code, be sure and credit me with my hard work!!"

But there is a **limit** to what developers will do for one another. When someone asked for even more help, the author of V3 replied:

> "As a general rule, I don't help other people code programs unless there is some profit for me. If you can offer some compensation, such as fair wages for a programmer (think anywhere from $30 to $300 per hour), then I will write your code for you."

The majority of watchers that comment on videos seem very eager to show and share their deep **appreciation** for the creators of the screencasts, as a watcher of R231 shared:

> "Mindblowing. Not enough words for it. This just convinced me to subscribe to watch all the code walkthrough episodes. Thanks a lot for the great work."

Finally, we found that developers use YouTube to **supplement their activities on other platforms** and to share screencasts. For example, interviewees spoke of how social media impacted their screencast creation process. Eight interviewees described how being active on other social media channels provided them with insights for their content. In this way, interviewees remained aware of programming-related news and events.

These insights are quite preliminary but point to a need for future work on how screencasts play a documentation role in a community of practice.

5.3 Limitations of Screencasts

It is important to note that there are limitations when using screencasts as a mechanism for providing documentation to developers. Our interviewees cited the lack of searchability as a barrier to finding relevant information on video. When searching for content, audience members are faced with the choice of watching the whole video and hoping that it contains the information they are looking for, or moving on to a different piece of content. For long videos, this is not efficient, so shorter videos may be preferred.

Video sharing platforms can help audience members find content by incorporating features like search, tags, and metadata supplied by the creator. Tags have been shown to be useful for helping developers discover and curate software knowledge on sites such as Stack Overflow (Mamykina et al. 2011). We observed YouTube screencast creators using metadata to help audience members find content, including linking audience members to additional resources such as source code and written tutorials. On RailsCasts, tags were used to organize screencasts into categories and each video included a written description.

The creation of screencasts also poses limitations. As was mentioned by the participants, recording and editing is not an easy task. For the screencast creator, once they have finished recording, it may be difficult to re-shoot and add missing information. Two methods that may help overcome this challenge include source-controlled projects to replicate code, and web-based development environments that quickly reproduce technical environments and dependencies.

Finally, a key limitation of using screencasts as documentation is that some forms of information are more easily digestible as text. Specifically, explicit and formal knowledge is easier to codify in a written form (Panahi et al. 2012), for example, coding style or where to place brackets for readability in source code. What screencasts support well is the sharing of experiences, knowledge of processes, and the practices involved in creating a specific object. Unlike written information, screencasts provide context into the technical environment of a developer and the ability to map source code to executable output.

6 Limitations of our Research

A primary limiting factor of this study is the generalizability from our sample of 27 YouTube videos (20 in Phase 1, 7 in Phase 3), 7 RailsCasts videos, and 10 interviews with YouTube screencast creators. Just considering YouTube alone, the potential population space contains thousands of screencasts. However, our findings were already well saturated after 20 screencasts in Phase 1 and 10 interviews—they revealed interesting insights and provoke future research into this phenomenon. Only two new codes emerged when we sampled a further seven YouTube and seven RailsCasts videos for analysis.

We relied on YouTube as the video source for the first two phases of 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. We next selected RailsCasts for our comparison analysis because of its professional level of quality and status within the Ruby on Rails community. This research only considered videos that were available from the two sites. It is possible that these factors (i.e., accessibility to content, storage, distribution, choice of search criteria) may have impacted our findings. However, as a first study, our intent was to explore how screencasts are used and to pave the way for future work. We discuss some of these ideas next.

7 Future Work

There are a number of directions to expand this work in future studies. The following themes focus on creating screencasts for developers and how developers use screencasts to support software engineering tasks.

7.1 Why and How Developers View Screencasts

While we know that developers publish content on media such as YouTube, we do not know how and to what extent developers use these videos to support development tasks. We were able to gain some insights from the comments, but we do not know how many people really watch these screencasts or how they use them. These questions should be explored through future studies.

7.2 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. It would be interesting to investigate tools used in professional settings.

8 Conclusions

Through this research, we provided an initial exploration into how and why developers create screencasts. In our study of 20 YouTube videos created by developers and 10 interviews with screencast creators (later supplemented by an additional 7 YouTube and 7 RailsCasts videos), 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.

Through analyzing a set of screencasts from the popular RailsCasts website, we compared the practices and techniques of a paid, managed, screencast platform to YouTube's open and *ad hoc* environment. Our analysis of multiple data sources led us to develop a list of best practices for screencasts and provided insights into the online developer community of practice.

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, Matthieu Foucault and Alexey Zagalsky for editing support and discussions about our research.

References

Adamic LA, Zhang J, Bakshy E, Ackerman MS (2008) Knowledge sharing and yahoo answers: Everyone knows something. In: Proceedings of the 17th International Conference on World Wide Web. ACM, pp 665–674

Adolph S, Hall W, Kruchten P (2011) Using Grounded Theory to Study the Experience of Software Development. Empir Softw Eng 16(4):487–513

Azer SA (2012) Can youtube help students in learning surface anatomy? Surg Radiol Anat 34(5):465–468

Biggerstaff TJ, Mitbander BG, Webster D (1993) The concept assignment problem in program understanding. In: Proceedings of the 15th International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, pp 482–498

Brooks R (1983) Towards a theory of the comprehension of computer programs. Int J Man Mach Stud 18(6):543–554

Capiluppi A, Serebrenik A, Singer L. (2013) Assessing technical candidates on the social web. Software, IEEE 30(1):45–51

Charmaz K (2006) Constructing grounded theory a practical guide through qualitative analysis pine forge press

Dabbish L, Stuart C, Tsay J, Herbsleb J (2012) Social coding in github: Transparency and collaboration in an open software repository. In: Proceedings of the ACM 2012 Conference on CSCW. ACM, pp 1277–1286

Dagenais B, Robillard MP (2010) Creating and evolving developer documentation: understanding the decisions of open source contributors. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, pp 127–136

Duffy P (2007) Engaging the youtube google-eyed generation: Strategies for using web 2.0 in teaching and learning. In: European Conference on ELearning, pp 173–182

Duncan I, Yarwood-Ross L, Haigh C (2013) Youtube as a source of clinical skills education. Nurse Educ Today 33(12):1576–1580

Ellison NB (2007) Social network sites Definition, history, and scholarship. Journal of Computer-Mediated Communication 13(1):210–230

Gubrium JF (2012) The Sage handbook of interview research: The complexity of the craft Sage

Guo PJ, Kim J, Rubin R (2014) How video production affects student engagement: An empirical study of mooc videos. In: Proceedings of the First ACM Conference on Learning @ Scale Conference L@S '14. ACM, New York, pp 41–50

Krippendorff K (2012) Content Analysis An Introduction to its Methodology Sage

Lankshear C, Knobel M (2010) Diy media A contextual background and some contemporary themes. *DIY media: creating, sharing and learning with new technologies*. Peter Lang, New York, pp 1–21

Lee M, Pradhan S, Dalgarno B (2008) The effectiveness of screencasts and cognitive tools as scaffolding for novice object-oriented programmers. J Inf Technol Educ: Research 7(1):61–80

Lethbridge TC, Singer J, Forward A (2003) How software engineers use documentation the state of the practice. Software, IEEE 20(6):35–39

Levy M (2009) Web 2.0 implications on knowledge management. J Knowl Manag 13(1):120–134

Lindvall M, Rus I (2002) Knowledge management in software engineering. IEEE Softw 19(3):0026–38

MacLeod L (2015) Code, camera, action!: How software developers document and share program knowledge using youtube. University of Victoria, Master's thesis

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

MacQueen KM, McLellan E, Kay K, Milstein B (1998) Codebook development for team-based qualitative analysis. Cult Anthropol Methods 10(2):31–36

Mamykina L, Manoim B, Mittal M, Hripcsak G, Hartmann B (2011) Design lessons from the fastest q&a site in the west. In: Proceedings of the SIGCHI Conference on Human factors in Computing Systems. ACM, pp 2857–2866

Mohorovicic S (2012) Creation and use of screencasts in higher education. In: 2012 Proceedings of the 35th International Convention on Information and Communication Technology. Electronics and Miroelectronics. IEEE, pp 1293–1298

Oud J (2009) Guidelines for effective online instruction using multimedia screencasts. Ref Serv Rev 37(2):164–177

Paek H-J, Hove T, Ju Jeong H, Kim M (2011) Peer or expert? the persuasive impact of youtube public service announcement producers. Int J Advert 30(1):161–188

Panahi S, Watson J, Partridge H (2012) Social media and tacit knowledge sharing: Developing a conceptual model. World Acad Sci Eng Technol 64:1095–1102

Parnin C, Treude C, Grammel L, Storey M-A (2012) Crowd documentation Exploring the coverage and the dynamics of api discussions on stack overflow. Georgia Institute of Technology Tech. Rep

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

Porter LV, Sweetser Trammell KD, Chung D, Kim E (2007) Blog power Examining the effects of practitioner blog use on power in public relations. Public Relat Rev 33(1):92–95

Räisänen T, Oinas-Kukkonen H (2008) A system architecture for the 7c knowledge environment. Information Modelling and Knowledge Bases XIX(166):217

Richardson WWH (2010) Blogs, wikis, podcasts, and other powerful web tools for classrooms, Corwin Press

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

Singer L, Figueira Filho FM, Storey M-AD (2014) Software engineering at the speed of light How developers stay current using twitter. In: International Conference on Software Engineering, pp 211–221

Snelson C (2011) Youtube across the Disciplines : A Review of the Literature. J Online Learn Teach 7(1):159–169

Storey M-A, Singer L, Cleary B, Figueira Filho F, Zagalsky A (2014) The revolution of social media in software engineering. In: Proceedings of the on Future of Software Engineering. ACM, pp 100–116

Storey M-A, Treude C, van Deursen A, Cheng L-T (2010) The impact of social media on software engineering practices and tools. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. ACM, pp 359–364

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

Swart J, Kinnie N (2003) Sharing knowledge in knowledge-intensive firms. Hum Resour Manag J 13(2):60–75

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

Treude C, Figueira Filho F, Cleary B, Storey M-A (2012) Programming in a socially networked world: The evolution of the social programmer. The Future of Collaborative Software Development:1–3

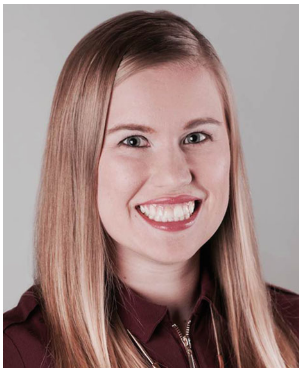
Treude C, Storey M-A (2011) Effective communication of software development knowledge through community portals. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. ACM, pp 91–101

Udell J (2004) Name that genre: screencast

Von Mayrhauser A, Vans AM (1995) Program comprehension during software maintenance and evolution. Computer 28(8):44–55

Wenger E (2000) Communities of practice and social learning systems. Organization 7(2):225–246

YouTube (2016) Youtube watch time optimization tips



Laura MacLeod holds a Master of Science from the University of Victoria.

Andreas Bergen is a PhD candidate at the University of Victoria.

Margaret-Anne Storey is a Professor at the University of Victoria.