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

by

Laura MacLeod
B.A., University of Victoria, 2012

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

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

by

Laura MacLeod
B.A., University of Victoria, 2012

Supervisory Committee

Dr. Margaret-Anne Storey, Co-Supervisor
(Department of Computer Science)

Dr. Yvonne Coady, Co-Supervisor
(Department of Computer Science)

Supervisory Committee

Dr. Margaret-Anne Storey, Co-Supervisor
(Department of Computer Science)

Dr. Yvonne Coady, Co-Supervisor
(Department of Computer Science)

ABSTRACT

Creating documentation is a challenging task in software engineering and most techniques involve the laborious and sometimes tedious job of writing text. This thesis 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.

This thesis presents a study investigating how developers produce and share developer-focused screencasts using the YouTube social platform. First, a set of development screencasts were identified and analyzed to determine how developers have adapted to the medium to meet the demands of development-related documentation needs. These videos raised questions regarding the techniques and strategies used for sharing software knowledge. Second, screencast producers were interviewed to understand their motivations for creating screencasts, and to uncover the perceived benefits and challenges in producing code-focused videos.

From this study a theory was developed describing the techniques used by developers in screencasts. This thesis also discusses YouTube's role in the social developer ecosystem, and presents a list of best practices for future screencast creators. This work lays the groundwork for future studies exploring how screencasts can play a role in sharing software development knowledge.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

There are many people who taught me important lessons throughout my graduate studies. I would first like to thank Dr. Margaret-Anne Storey and Dr. Yvonne Coady for their constant support and guidance, while I was a member of their labs. I am a better researcher and person because of the opportunities you gave me, and I feel very lucky to have worked with both of you.

I was lucky enough to be surrounded by an amazing group of people at the University of Victoria. I would like to especially thank Cassandra Petrachenko for her vicious wit and superb editing skills, not just on this thesis but on many other projects. Elena Volishnakova, thank you for being my lab buddy and for making me laugh. The CHISEL lab was a great academic environment and I consider myself lucky to have been a part of it. I will miss the Modsquad and Chisel-ers.

Finally, I owe a big thanks to Andi Bergen who gave his time and support in analyzing the screencast data presented in this thesis.

DEDICATION

I dedicate this thesis to my family.
You continually amaze me.

Chapter 1

Introduction

Software engineering is an "intellectually demanding" task and how developers understand and comprehend programs has been thoroughly studied in computer science [28]. To support cognitive tasks, today's developers use a variety of tools, including those based on mass participation via the Internet.

The Internet helps developers search for, locate, and explore information [10]. This information is contributed by people from around the world. Specifically, developers use the Internet to access social media, which allows them to collaborate with each other [76], find answers to their questions [81], and participate in online communities [27].

Previous work has explored how developers use social media platforms to support their work, including how developers use GitHub to collaborate on projects [27], Twitter to follow thought leaders [70], Stack Overflow to ask and answer questions [81], and profile aggregators to manage their online identity [71]. This has led to the rise of the term "Software Developer Ecosystem" [75].

From the literature on social media and software development, we know that developers participate in diverse ecosystems, using a number of social media platforms. What has received little attention in the literature is the role of screencasts in this ecosystem. From the array of videos on YouTube, we know that developers use social media platforms to share screencasts.

Screencasting is not a new phenomenon. In 2004, John Udell asked his blog readers to submit terms for "making movies of software" [83]. From entries such as "vidcast" and "software movie", Udell chose the term "screencast". In his blog posts announcing the term screencast, Udell wrote, "I continue to be fascinated by this medium. The ability to capture, narrate, and share software experiences ... enables

an important mode of communication that we've barely begun to exploit" [83]. Despite this enthusiasm, little work has been done to understand the role of screencasts in software development. Equally lacking in the literature is work exploring how YouTube contributes to software development.

To explore screencasts and YouTube from a software engineering perspective, this thesis considers how developers create screencasts and share their creations with other developers. To explore this topic, screencasts hosted on YouTube were analyzed and interviews with screencast creators were conducted.

This work aims to uncover why these screencasts exist, to understand the processes that lead to their creation, and the experiences of the developers who made them. By understanding how developers share knowledge, we can better support the creation of new tools and guidelines for developers seeking to create screencasts.

This thesis explores the processes and techniques used to create and share 20 screencasts and interviewed 10 screencast creators for additional insights. With the assistance of a collaborator, a content analysis approach was used to analyze the interview data, and we relied heavily on grounded theory methods to analyze the screencasts[1].

Our analysis led to the emergence of four research questions:

1. What kinds of program knowledge are captured in screencasts?
2. What techniques do developers use to explain code in screencasts?
3. Why do developers create code screencasts?
4. How do developers produce code screencasts?

The resulting theory illustrates the techniques used by developers in screencasts to share knowledge. The theory is situated using related work on screencasts, YouTube, and knowledge sharing practices.

1.0.1 Thesis Organization

This thesis is organized as follows. In Chapter 2, the literature on YouTube, screencasting, and knowledge sharing is surveyed. The chapter also presents literature on how developers use social media platforms.

[1] Andreas Bergen, a current graduate student at the University of Victoria, assisted with the screencast coding and interview transcript analysis.

In Chapter 3 the methodologies used to collect and analyze data are outlined. Chapter 3 also discusses my constructivist epistemological foundations.

Chapter 4 presents the findings and answers to the research questions. Data from the interviews and screencasts is integrated into the resulting theory, as outlined in Chapter 5. In Chapter 6, a list of best practices for future screencast creators is presented along with a discussion on YouTube's place in the social developer ecosystem and challenges to screencasting.

Limitations and threats to validity are outlined in Chapter 7, followed by future work in Chapter 8. Finally, the thesis concludes in Chapter 9. Additional documents are found in the Appendices, including the coding handbook and interview questions.

Chapter 2

Background and Related Work

Software developers live and work in a social space. Recent work has termed the phenomenon of incorporating social media into software development, "the social programmer ecosystem" [75]. The social programmer ecosystem describes developers building support structures, "around content, technology, media..." [75] Previous work by Storey et al. has explored how developers use multiple social media channels to support their personal and professional goals [75]. YouTube fits into this system by allowing developers to share video content, centered on the technologies and tools that appeal to their programming interests. This chapter presents literature about YouTube, screencasts and knowledge sharing practices in social programming environment.

2.1 YouTube

Officially launched in 2006, YouTube is a social video sharing site [19]. With over 100 hours of video uploaded every minute [87], YouTube represents a rich record of information from every corner of the globe. YouTube not only lets users upload video easily, it also provides them with the mechanisms for providing feedback and finding material.

As a knowledge sharing resource, YouTube provides content and social networking mechanisms. It allows people to establish an identity and thus enables connections between users. Users build their identity by sharing videos, following other users and commenting on videos.

Through video, users share their experiences, which Wenger claims encourages dialog among communities [86]. The experiences captured in YouTube videos contribute to and provide insight into the cultural practices of a community. According to Burgess and Green, these videos serve as a collection of social norms and opinions [13]. Because of its low barrier to entry, YouTube has impacted how people present information to the public [39]. For the software development community, YouTube represents a medium that developers can use to share programming practices.

2.1.1 YouTube and Learning Literature

Previous work has investigated the use of YouTube in education. The studies referenced below provide examples about how it can be used to support learning.

From the previous work of Duffy, we know that students like YouTube for delivering educational content as it provides a, "user guided experience" [31]. Mullamphy et al. found that students claim they enjoy using YouTube videos to learn at their own pace [59]. Other studies, like the work of Duncan et al., have found that YouTube encourages students to explore topics on their own [32]. YouTube has also been used in the medical field to teach medical students clinical skills [32] and about rare medical cases [6]. These videos allow students to replay film to better understand how to apply techniques, and to observe rare, or interesting, cases.

2.2 Screencasts

While YouTube is used to broadcast a variety videos, ranging from music videos to gag reels, this thesis focuses on screencasts, or videos that show a person's computer screen. The term screencast is also used to describe, "making videos of software" [83].

Previous research has explored the techniques and strategies used in screencasts for formal, classroom learning [79]. Other work has put forth best practices for teachers looking to incorporate screencasts into their curriculum [61].

Sugar et al. explored educational screencasts to identify common elements and the techniques used in them [79]. From their research, they created a list of elements and strategies used in educational screencasts. Other work has provided guidelines and best practices for educators who wish to create screencasts. For example, the work of Oud et al. provides a number of best practices for creating screencasts [61]. Their

work also suggest including interactive components or activities for the audience. Their work highlights the importance of feedback between the screencast creator and the audience.

Other researchers have found that screencasts help students learn to do tasks more effectively. For example, Tempelman-Kluit found that an audio/video tutorial improved students' learning [80]. Work by Kocejko used video tutorials to teach seniors how to perform basic computer tasks, thereby helping students learn to do tasks more effectively [46].

In classes with a laboratory component, these videos serve as a way to prepare for the lab and troubleshoot during the task [34]. Fortino and Zhao noted that students in a laboratory setting seem to spend more time troubleshooting their equipment than actually performing the given task [34]. Doering and Mu described how video tutorials allow students to pause or replay the video, which provides students with the ability to walk through the steps at their own pace, instead of relying on an instructor [30].

This previous work demonstrates that there is an academic interest in understanding screencasts. The literature has explored how screencasts are made [57], what techniques are used in them [79], and best practices for creating them [61]. This thesis explores these issues, but from a software development perspective. This study aims to support software developers by understanding how they can be aided by screencasts.

The studies discussed above explored screencasts as a standalone artifact. In this thesis, screencasts are explored in the context of a social media environment. This work builds upon the previous research on screencasts to explore how it contributes to the social developer ecosystem.

2.3 The Social Programmer Ecosystem

Today, developers work in a social ecosystem. Previous literature describes how social media has led to the rise of the social programmer who participates in, "ecosystems around content, technology, media and developers" [75]. This ecosystem is made up of artifacts and resources accessible to developers via the Internet.

The following sections present how this social ecosystem impacts software development, and specifically, how it affects the developer. Through the use of social media, developers share their experiences and knowledge with others. This section will ad-

dress how and why developers share knowledge through social media platforms, with a focus on software development tasks.

2.3.1 Developer Identity

As defined in previous work by Storey et al., the term "social programmer" refers to a programmer who actively participates in online communities [75]. These developers embrace crowd-sourced, socio-technical content, accessed through the Internet [75] — this content comes in many forms, including blog posts and comments, and social media platforms give developers an outlet for contributing and accessing the information.

Dabbish et al. described how developers participate in social ecosystems, which leads to the development of an online identity that becomes associated with the developer's knowledge and skills [27]. It affords them informal membership to many communities of practice, which Wegner describes as being built around members with common interests and goals [86]. Other work by Capiluppi et al. suggests that users will use this identity to evaluate the developer and shape their interactions with them [15].

An online identity motivates developers to curate their online persona and how they present themselves. It motivates developers to continue their participation and share their experiences through social media [64]. As one researcher put it, "social media has rapidly changed how programmers advertise their skills and how they manage their time coding" [82]. This identity has also been found to influence what projects developers choose to work on [27], and potential employers may use an online identities to assess potential candidates [15]. Because the identity is open to the public, developers are conscious of their work and persona.

2.3.2 Awareness

Social media has been found to not only help developers establish identities, but also create awareness among communities about trends and cultural norms [27]. It gives users the ability to share new developments instantaneously and inform community members [76]. The work of Wegner described how these communities can be used for members to determine where knowledge is located [86].

In the work of Singer et al., researchers explored how developers use Twitter to follow trends, and stay aware of developments [71]. They found that developers

use Twitter to follow thought leaders and learn about the release of tools and languages [71]. In this way, they use Twitter for "just in time awareness" about the status of a technology.

2.3.3 Knowledge Foraging

Developers also take advantage of social media to locate knowledge for software development tasks. This has been found to include using social media to locate code snippets and code examples [64]. Through the use of search, developers are able to find and incorporate code snippets on the fly. Brant et al. studied this phenomenon and called it, "opportunistic Web foraging" [10]. As another research paper described it, with the power of search, "reusable code snippets, introductory usage examples and pertinent libraries are often just a Web search away" [82].

An example of this is Question and Answer sites, which provide developers with outlets to ask questions and find answers. Notable Q&A sites include Yahoo Answers and Stack Overflow [81]— with over 30 million unique users per month, Stack Overflow is an example of developers using social media to share insights, and ask questions [67]. As the literature has shown, developers use social media platforms to support their software development processes and search for knowledge.

2.3.4 Collaboration

These social media platforms can also be used to promote collaboration amongst developers. For example, Singer et al. describe how social media allows developers to connect, monitor and publicize their activities in a public space [70]. By doing this, other developers can access their work, and draw from their experiences.

For example, GitHub hosts the projects of over 3.5 million users [35]. GitHub represents a common space for developers to share their work, and provides support for users to connect, communicate and contribute to projects. The transparency of this social space has been found to support collaboration through increased awareness [76].

Social media platforms have also lowered the barrier for participation [70]. Previous work has shown how developers are able to engage in public conversations on Twitter with those who have similar interests [71]. The work of Singer et al. finds that these public conversations allow users to promote their ideas and connect with strangers [71].

2.3.5 Sharing Knowledge

Through social media participation, developers both consume and produce content. In this thesis, YouTube screencasts are explored as a method to capture developer knowledge and experiences. Through channels in the social development ecosystem, developers find and contribute information. For this reason, it is important to understand how knowledge is shared in these spaces. This includes knowledge that is explicitly stated in screencasts, as well as knowledge that is tacitly conveyed through the practices shown in screencasts.

The rise of the Internet has led to the rise of, "communities of practice" — Wenger described these as communities that form around shared interests and common goals [86]. According to Wenger's social learning perspective, one belongs to a community through methods of engagement, imagination and alignment [86]. Within these communities, participants place high value on the exchange of knowledge [55].

One of the impacts of communities of practice is that developers willingly contribute their knowledge with no expectation of direct compensation [51]. In this way, it has been suggested that social media has created a culture dedicated to sharing and encoding knowledge [2]. Developers use these platforms to share their expert knowledge, for the sake of contributing to a larger community [70]. Work by Levy suggests that developers share their experiences on social media for "altruistic" means [50].

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 [64]. It has been suggested that social media support users in locating resources [33], which in turn contain this explicit knowledge. Explicit knowledge can be combined with other forms of knowledge and ultimately internalized by an individual [60].

Tacit knowledge can be supported through informal mentoring, the sharing of experiences, observations, discussions, and trust within a social media platform [63]. Dabbish et al. describe how developers are able to establish expertise and build trust by forming an online identity [27]. Transparency and awareness in social media communities supports the sharing of experiences and observation by other developers. With knowledge foraging activities, developers are able to reflect on the experiences of others. The openness of the platforms provides common spaces for discussion to occur.

While participating in communities of practice, members are constantly engaging in what Wenger described as "knowledge production, exchange and transformation" [86]. Members share their experiences, which Hemetsberger and Reinhardt demonstrate leads to the sharing of tacit knowledge [42]. Wenger suggests that sharing these experiences allows social norms and values to develop [86]. Through these communities, developers can follow the work of others, while simultaneously contributing their own work to the collective knowledge base. It allows developers to engage in "serendipitous learning", where they come across useful materials by accident [71]. These activities create social ties between developers, which in turn supports knowledge sharing activities [47].

This section has listed some of the ways in which the social developer ecosystem provides support to software developmental practices. The extensive participation seen online shows that developers are comfortable using social media to locate, contribute and consume knowledge. These activities support developers through the establishment of online identities. Through participation and sharing experiences, developers share tacit and explicit knowledge. The transparent nature of social media supports developer awareness in their field, and helps them reflect on the experiences of others.

In this chapter, previous work on YouTube and screencasts has been presented. It has also shown how software developers use social media to support software development tasks. They participate in social media ecosystems to exchange knowledge and to find information. It allows them to connect and follow other developers, which leads to collaboration. Through social media, developers are able to share their experiences, locate knowledge and establish an identity.

Chapter 3

Methodology

This work began with an analysis of screencasts collected from YouTube. This was followed by interviews with developers who had created screencasts for YouTube. This was done to gain further insights into the screencast creation and socialization process.

From the screencast analysis, four research questions emerged. These questions address how and why developers share knowledge through YouTube, including the techniques used in screencasts, the motivations developers presented for creating screencasts, and the processes used to create them.

This chapter presents a background on the methodology, and lays the foundations for the epistemological position taken in this work. The chapter also outlines the procedures that were used to collect and analyze the data in this study. The findings of this analysis are presented in Chapter 4.

3.1 Philosophical Assumptions

In this thesis, I subscribed to a constructivist epistemology. Epistemologies provide frameworks for the philosophical assumptions we make about knowledge and research [25]. The following section outlines the epistemological assumptions I held in this work, regarding how we can observe the world around us and construct knowledge.

3.1.1 Constructivism

According to Creswell, constructivism holds that the world is a social construction, interpreted by humanity [25]. Specifically, that reality, "is an evolving process, concrete in nature, but ever changing in detailed form" [58]. Therefore there are many ways of knowing the world.

Drawing on the work of Morgan and Smircich, constructivism assumes that people influence, and are influenced by, their environment through interactions [58]. Morgan and Smircich state that people assign subjective meanings to their actions, leading to many different interpretations of a shared experience. While conducting research, the researcher constructs meaning against their understandings and interpretations of the world [68].

3.1.2 Methods

For this research, qualitative methods were used. In relying on a grounded theory approach, the researcher is able to ground their findings through the experiences of others [25]. For this thesis, I conducted interviews and analyzed the results. My interpretation of the results formulates the findings of this work.

The researcher is never a blank slate, but by acknowledging their biases, they can become aware of ideas that may influence their work [78]. Based on recommendations from the literature, my biases and assumptions are outlined in the next subsection. By outlining my assumptions and biases, I aim to provide context for my approach and methods. These assumptions will be addressed as limitations to this study in Chapter 7.

3.1.3 Personal Reflections

There are numerous biases that the primary researcher brought to this work. I present these assumptions and influences as a way of reflecting their impact.

As the primary researcher, I have a history of education in the social sciences, specifically in political science. From this personal experience, I bring the world-view that knowledge is multifaceted and open to interpretation. People have an individual set of beliefs that they use to understand the world around them. I hold that there are many interpretations that influence the understanding of a phenomenon.

Originally, this work began as an exploration of onboarding new contributors in open source software projects. I was curious about how open source projects provided information to new contributors. Embedded in this line of thought was the question, "How do developers share knowledge with each other?" From this, I was drawn to video in open source projects. This led to an exploration of videos describing code. I was further drawn to the role of social media and screencasts hosted online, which finally led us to the exploration of YouTube screencasts.

I have previously studied the impacts of social media on software development in other work [52]. Therefore, I had existing knowledge that developers use social media to promote themselves and find information.

A colleague assisted me in coding the screencasts and analyzing the interview transcripts for this study. We both have knowledge of programming fundamentals, which allowed us to analyze the screencasts. We both have experience creating programs and understand common programming concepts, such as data structures and algorithms. This background shaped and supported our analysis.

3.2 Grounded Theory

Grounded theory is a methodology which requires the researcher to explore their data with the goal of generating a substantive theory [77]. Specifically, the methodologies of Charmaz were used to carry out this thesis [17]. The following section provides a background on grounded theory and the methods that contribute to this approach.

3.2.1 Foundations of Grounded Theory

As a qualitative approach, grounded theory has been described as being especially well suited for situations where there is little existing work on a subject [26] —it helps answer questions about, "what is going on in an area" [56]? Grounded theory allows the researcher to explore the data to discover meaning from it using an inductive approach [22]. In this way, the researcher, "investigates the actualities in the real world" and can devote their analysis to issues seen in the data [7].

This does not imply that grounded theory does not provide the researcher with a set of guidelines. On the contrary — the work of Glaser, Strauss and Corbin has provided researchers with rich and systematic frameworks for applying the methodology [1]. In a grounded theory method, both data analysis and theory construction

occur in a, "constant comparison" [77]. Because of this, the researcher is consistently checking their analysis of the data against new and emerging ideas. Central to the practice of grounded theory are techniques such as open coding, memoing and theoretical sampling, with interviews being a common data source [26].

According to Creswell, the result of a grounded theory study is a theory that describes an observed or experienced phenomenon [26]. Creswell notes that with grounded theory, the researcher is not trying to create a broad, all encompassing theory. Rather, they are trying to make sense of the data at hand. According to Charmaz, these "middle range" theories are meant to address the issue at the level the researcher is engaged with, as opposed to a theory that explains the whole population [17]. In her writing, Charmaz advocates that the researcher not try to be a "hero", but rather that they formulate a theory which is appropriate for the sample being observed [17]. The analysis in this thesis is limited to to a sample of screencasts and interviews. These allowed the development of a theory that explains the techniques used in screencasts and situates them in social factors.

3.2.2 Ideological Camps

Glaser and Strauss first introduced the grounded theory technique in the book *The Discovery of Grounded Theory* [36]. The two authors developed the methodology while researching hospital patients, as they reportedly wanted to better understand the process of dying [37].

According to Charmaz, subsequent work by Corbin and Strauss created a divergent approach to grounded theory [17]. The two approaches are described as having different epistemological viewpoints, with Glasser being described as a strong positivist [44], and Strauss as a pragmatist [56].

Strauss went on to work with Corbin to develop what has been described as a more pragmatic school of thought [45]. Glasser's main critique of Strauss and Corbin's work is that the rigid procedures force data into preconceived categories, as opposed to letting ideas emerge [17]. A critique of Glaser's work is that it is too open, and does not provide enough guidance to the reader [45].

Charmaz continued the grounded theory tradition through her exploration of chronic illness. Charmaz differs from Glasser, Strauss and Corbin in her epistemological approach to grounded theory [17]. Charmaz's school of thought founds itself in ideas from Glasser and Strauss' earlier work, advocating that grounded theory is

a flexible strategy for exploring data [17]. Her approach holds that data and theories are not discovered, but constructed. It is not surprising then that Charmaz has been described as a constructivist [26]. We focus our approach on Charmaz's writings, specifically her 2006 book *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis* [17].

3.2.3 The Place of the Literature Review

The literature review is a controversial subject in grounded theory. Different ideological camps have varying opinions about where it fits in the grounded theory approach. The work of Kelle explains Glaser's approach, which dictates that there must be no literature review when applying a grounded theory method [45]. Strauss and Corbin criticize this and take the stance that the literature review may happen as needed [45]. Charmaz dedicates a section of her book to explaining these different traditions, and putting forth her own opinions [17]. She write that the literature review should not limit or "stifle" the work of the researcher, and fear of the review biasing the final grounded theory is not an excuse for ignoring the literature [17]. It is also important that the literature review is done in a way that does not close off the researcher to ideas from the data.

The literature review was performed throughout this study. Being a graduate student, there was no way for me to begin this work without some exploration of the state of the art. The majority of the literature review was conducted towards the end of the study, well after themes and codes had been drawn from the data. Initial research on program comprehension and onboarding topics was conducted at the beginning of this study. Literature on knowledge sharing, YouTube and screencasts was reviewed after the data had been collected and analyzed. Key pieces of literature, such as educational work on the elements and strategies of screencasts, were discovered towards the end of this thesis. In this way, connections were drawn between the observed phenomena and existing work in the field.

3.2.4 Why Grounded Theory?

Grounded theory was chosen because there is little existing work on screencasts and software developers. As Creswell explains, qualitative research aims to collect, "data in a natural setting sensitive to the people and places under study" [24]. This thesis poses questions of "how" and "why" by seeking to describe the experiences of devel-

opers in relation to screencasts. By using an inductive method like grounded theory, the researcher is free to explore the data while looking for answers to these questions.

The Charmaz approach to grounded theory was chosen for this thesis because of the resources available to the researchers, and because of its constructivist foundations. The Charmaz tradition also provides a base of literature with guidelines and questions for conducting grounded theory research [26]. In this way, the Charmaz approach provides guidance to those looking to undertake grounded theory analysis. Because of the subject area being explored, my epistemological beliefs, and the type of data being collected, a grounded theory approach was considered to be the best approach for this thesis.

3.2.5 Grounded Theory in Software Engineering

Grounded theory is well established in the study of software engineering. It is applied in software engineering research to understand, "social and cultural issues" that emerge between people and computers [21]. The following paragraphs provide examples of grounded theory being used to study software engineering concepts. From these examples, we can see how the method is used to study the dynamics of people working with software. For example, Coleman and O'Connor explored the software improvement practices of Irish software companies using a grounded theory methodology [21]. They used semi-structured interviews to generate a theory about how management implements software improvement practices. Adolph et al. used grounded theory to answer the question of, "How do people manage the process of software development?" [4] This study put forth a list of recommendations, centered on improving communication and negotiations among software development teams [5]. In both cases, grounded theory allowed the researchers to collect data from a variety of resources, interview direct participants about their experiences and put forth theories based on their observations.

Grounded theory has also been used in a software engineering context to understand the user experience. For example, the work of Pace used grounded theory to understand flow, "[a] state of consciousness... when involved in an enjoyable activity" [62]. He looked at how users experience flow while completing tasks on the Internet. This was a scenario where a lab study would not have been appropriate. Another example of grounded theory is the work of Ploderer et al., who examined online and offline relationships facilitated by social media [66]. In this study, the

researchers looked at a "passion-centric" website for people who were bodybuilding enthusiasts. Finally, DiMicco et al. used grounded theory to explore how people use social media at work [29]. They were able to uncover how and what information employees share at an enterprise company. They showed how employees' motivations for sharing information on within their professional social network differed from traditional social media use.

This section has outlined how grounded theory has been used to explore software engineering practices. This thesis explore the experiences of developers who form relationships through social media by sharing content. From the examples provided, one can see how other researchers have used grounded theory to explore how social media mediates relationships and tasks.

3.2.6 The Sequence of Grounded Theory

The application of grounded theory does not occur in a sequential fashion; to present this research methodology as a linear process has been described as being misleading to the reader [78]. According to Jones and Alony, one does not begin grounded theory with formed research questions in mind, but with areas of interest to explore [44]. This study presents the research questions as fully formed, when in reality, they underwent multiple revisions.

A key principle of a grounded theory is using an inductive analysis to study the data. This means reviewing the data multiple times to identify constructs and ideas [26]. Creswell describes the grounded theory researcher as zigzagging between data collection and data analysis [26]. This is contrary to other methods, where the data collection and analysis phases are kept distinctly separate.

In collecting the data for this thesis, ideas were built up through the use of a number of techniques described in the following sections. The results of grounded theory work are presented to the reader after iterations of analysis and refinement. It is important to note that the following sections have been presented in a linear fashion for the sake of clarity for the reader. Using a grounded theory method, various stages of this study overlapped.

Throughout this work, we constantly switched between data collection and data analysis; Figure 3.1 provides a basic visualization of this progression. As can be seen from the diagram, collecting and analyzing the screencasts was an iterative process that my collaborator and I went through numerous times, in order to refine our codes.

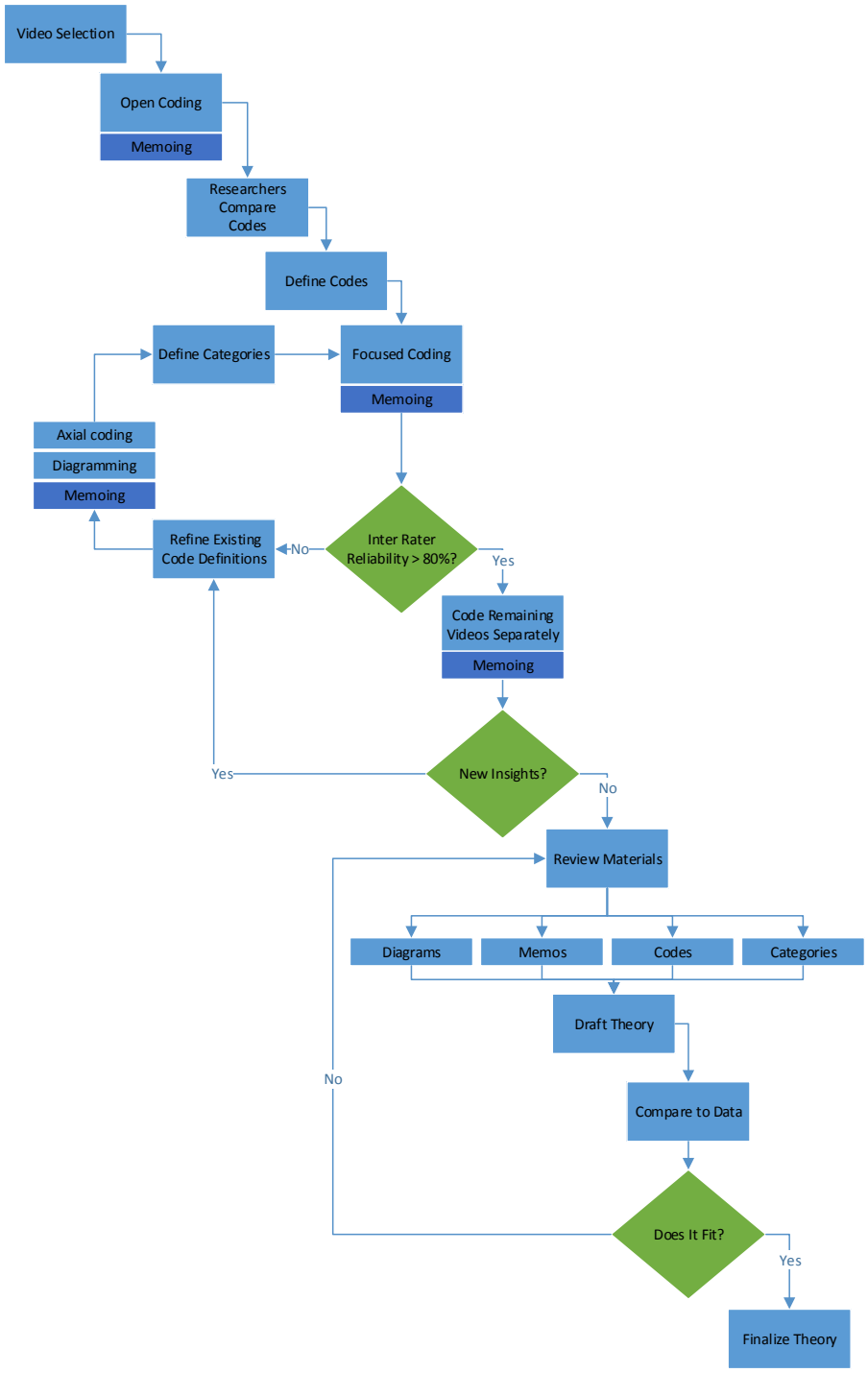


Figure 3.1: This chart shows how the researchers used various data collection and analysis techniques to develop the findings of this work. This research path is based on a grounded theory methodology.

An example of the "zigzag" in this work is that only after the screencast analysis was underway, did questions about how developers' motivations for creating these artifacts emerge. Later, while doing interviews, interview data was collected and analyzed at a near simultaneous rate. Again, codes and themes were mapped and defined while data collection was ongoing.

3.3 Research Questions

From the analysis of the data, four research questions were formed that provide clarification for the goals of this research. As previously mentioned, screencast data was used to answer research questions one (RQ1) and two (RQ2), while interviews were used to answer research questions three (RQ3) and four (RQ4).

3.3.1 (RQ1) What Kinds of Program Knowledge Are Captured in Screencasts?

From the literature, we know that some developers rely on screencasts (and podcasts) to learn technical information [75]. This led to the question of what high-level goals lie behind the screencasts of code implementations. This exploration focused on what is shown in these videos and what problems the creators are trying to solve. These goals were determined by analyzing a sample of videos hosted on YouTube.

3.3.2 (RQ2) What Techniques Do Developers Use to Document Code in Screencasts?

This work focused on videos that showed a completed project or code snippet. The benefit of these videos is that they present working examples instead of content aimed at teaching the fundamentals of programming. The wording of this research question was iterated on a number of times to reflect the themes emerging from the data. An early form of this research question was, "How do developers describe code in these videos?" It was later changed to, "What techniques do developers use to explain code in screencasts?" before setting on the final question presented here. This iteration was done in an attempt to be open to the data and follow the guidelines of Charmaz [17].

3.3.3 (RQ3) Why Do Developers Create Code Screencasts?

From analyzing the screencasts, one could see that developers would often produce more than one screencast. They also interacted with their audience via YouTube comments. These were the first indicators of social activities and prompted questions about developers' motivations for creating screencasts. The question of why developers create these screencasts also addresses how screencasts fit into the larger developer social ecosystem.

The question's phrasing was also iterated upon and at one point was, "Why do developers make code walkthrough videos?" The wording, "walkthrough videos" reflects the focus (at the time) on the developer guiding the audience through the code. The interviews conducted in Phase 2 of this study explored what motivated developers to create screencasts, share them on YouTube, and the feedback they had received from their audiences.

3.3.4 (RQ4) How Do Developers Produce Code Screencasts?

Finally, the actual practices of creating these videos were explored. The question of how developers produce screencasts allowed me to focus on the tools and processes that they used. This question was designed to help investigate the current state of practice. By investigating the processes developers' use, common techniques were identified, as well as areas of friction expressed by the interviewees.

3.4 Phase 1: Screencast Analysis

With the research questions outlined, this section presents the steps taken to gather and analyze the data. The findings from this data are addressed in Chapter 4. The following sections outline the procedures and provides literature on the specific methods we used.

3.4.1 Screencast Selection

This study began with the gathering a sample of videos from YouTube. YouTube videos were chosen because they are openly available, multimedia resources, which convey technical information. They tell the user how the code works and provide insight into the structure of the program. These artifacts contain socio elements,

such as attitudes or cultural values, and they are a persistent, computer-based media item that also serve as documentation. YouTube allowed us to explore these artifacts as they relate to social media and software development.

In order to select videos for my sample, a list of inclusion criteria was created. This list included:

1. **The video was available on YouTube from November 2013 until March 2014**, which was the data collection phase of this study.

2. **The video presented a completed code base or code snippet**. The scope of this research did not include tutorial screencasts, or screencasts that teach the fundamentals of programming. It is assumed that the audience has some preexisting knowledge about programming concepts. Acquiring knowledge through code walkthroughs is applicable to other software engineering tasks. Brooks explains that it also applies to maintenance, debugging, testing and alteration of software [12].

3. **The video description or title contained a form of the term "Code Walkthrough" or "Code Tour"**. The videos collected were limited in that they had to contain the words "Tour" or "Walkthrough" in the title or video description. These terms were chosen as the most efficient way of finding videos that met our research goals. In collecting the videos, I searched YouTube for videos that contained either of these words plus additional descriptors such as "Program", "Software", "Code", "Java", "Python", "Ruby" and "Unity". These descriptors focused the results on videos about software material. The terms "walkthrough", "tour" and subsequent descriptors had an impact on the types of videos chosen for this thesis. The terms were chosen to focus our search on videos that showed completed code and because the terms imply guidance through a project.

4. **The video contained an audio description with screencapture of the narrator's computer**. In this thesis, the term screencast is used to describe the video under analysis. With this in mind, it was required that the videos analyzed show the narrator's screen and relevant code. All of the videos had an audio narration component, as well as screen-captured video.

Screencasts that focused on explaining programming fundamentals were excluded, as screencasts aimed at teaching the basics of programming have a very different

purpose. Rather, this work focuses on 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.

In order to select videos for the set, I went through the search results and watched the first few seconds of each video, to see if it met the requirements. Videos that did not meet the criteria were excluded from the set. The data set was capped at 21 videos, when my collaborator and I deemed our observations as being saturated. A video was removed from this set later, after it was found to not meet the initial criteria. The final data set represented over eight hours of footage. Though this represents a fraction of the possible screencasts hosted on YouTube, my findings were well saturated using this sample.

Table 3.1 provides further details on the characteristics of the screencasts we selected. The topics of these videos range from Arduino projects to Django, the Corona SDK and Unity. Note that the videos range in length: the longest is over 90 minutes and the shortest is 3.

3.4.2 Program Knowledge Analysis

To answer RQ1, I watched the screencasts. I recorded who made the screencast (a company or a person), as well as details about the goal of the screencast. From this, it was determined why developers created the screencast, and what knowledge they were trying to share. Through analyzing and discussing our records, RQ1 was answered.

3.4.3 Open Coding

Charmaz suggests that the researcher ask, "What is this data a study of?" [17]. The screencast data is a study of code being presented in video format by developers, and more specifically, videos of a narrator explaining a program that they are familiar with. With this in mind, the video analysis began using an open coding technique, as suggested by Charmaz [17]. Open coding involves reviewing the gathered data systematically with the goal of discovering common trends or ideas [4]. Using grounded theory, my collaborator and I created codes, and wrote memos to supplement our thinking while analyzing the data.

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 3.1: The YouTube videos analyzed. All YouTube video URLs take the format of "https://www.youtube.com/watch?v=" plus the URL Link. View and comment counts as of Nov 2013. *View counts unavailable for Google Hangouts.

During the initial stage, we coded for what we saw or heard in the videos. We wrote memos as needed, noting similarities and repetitive processes shown in the videos. At this stage, Charmaz suggests that the codes should be precise to preserve meaning and clarity [17]. Using paper and pencil, we first coded a set of videos separately. After this, we met to compare findings and begin developing codes.

From our initial codes, we began the process of creating a coding book to outline definitions. This set of codes are examples of what Charmaz terms "focused codes" [17]. At this point, the researcher moves a step away from coding every occurrence and begins seeing larger trends within the data.

For this stage, we used Excel spreadsheets to code the data. The first column of the spreadsheet contained time stamps, increasing incrementally by five seconds. Figure 3.2 shows an example of a coding spreadsheet. We decided to use five-second increments, because keeping track of events to the second proved to be too time consuming. The second, third and fourth columns were used for codes and memos as needed. This allowed us to quickly combine and compare the coding done by each researcher.

Figure 3.2 shows an example of our coding method. The example shows code C1 starting at 0:00 and continuing until 0:30. A second code, B1, begins at 0:30. Codes were given shorthands to make coding easier for the coders. As we continued to explore the data, we combined codes and dropped others. The final codes are presented in Chapter 4.

3.4.4 Coding Book

To formalize the codes a coding book was created. An initial set of four videos was coded over four sessions. In each session, two of our researchers separately coded two videos. Multiple sessions allowed us to refine definitions in the codebook. The codebook served as documentation for the formal definitions of codes during the video analysis phase. Each entry in the coding book was given a descriptive title, a formal definition, an example, and space for any further notes or thoughts from the researchers. The final version of the codebook can be found in Appendix A.

The book was used to note the methods of interpretation by the researchers during the initial coding phases. Over time, codes were combined or removed based on our analysis. The book also allowed us to track disagreements or misunderstandings over time and this helped clarify the definition and scope of the codes.

Inter-rater Reliability

Inter-rater reliability refers to the accuracy with which a group of researchers code the same body of evidence [17]. Based on guidelines for conducting qualitative research, the level of inter-rater reliability for coding was set at 80% [53]. An agreement formula was used to calculate the level of inter-rater reliability [72]. Figure 3.3 shows this formula. All the instances where the coders agreed were calculated against all the instances where they disagreed.

Other statistical approaches were considered, but decided to be impractical based on the format and goals of our coding process (discussed further as a limitation in Chapter 7). Inter-rater reliability was used to provide guidance and measure our coding agreement process.

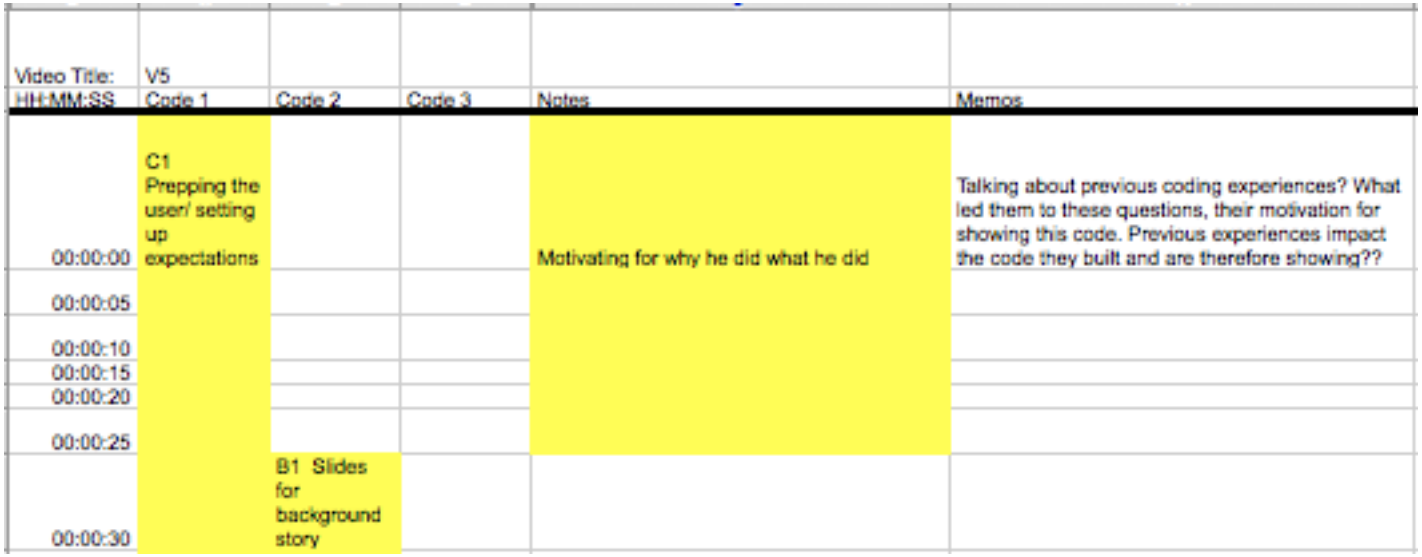
There are acknowledged limitations to using a simple percentage agreement calculation. For example, the high occurrence of one variable that the coders agree upon, may increase the resulting percentage, thus hiding more infrequent but consistent areas of disagreement [72].

It is uncommon in the grounded theory literature to use inter-rater reliability. This is because inter-rater reliability is a quantitative approach. In this thesis, I argue that it was appropriate to use a calculation because it did not constrict our coding. In fact, it forced us to have meaningful discussions about our coding and the assumptions we made about the codes. Any disagreements were noted in the coding book and these led to more refined codes. Noting where we disagreed provided insights into the assumptions made while coding. Once the inter-rater reliability threshold was reached, we coded the remaining videos separately. It took two sessions to pass this threshold.

3.4.5 Memoing

Memoing is the technique of writing down ideas while analyzing data [25]. These memos may take the form of paragraphs or short ideas.

In the screencast analysis, memos were written in the Excel spreadsheet next to the applicable codes as the researcher coded. According to Charmaz, these notes act as a way of capturing the researcher's thinking in the moment of analysis [17]. Memos were made asking questions about what the presenters were doing in their videos and noting common trends. For example, in Figure 3.2 the researcher wrote a memo asking questions about the narrator's motivations for creating the video.



Video Title:	V5				
HH:MM:SS	Code 1	Code 2	Code 3	Notes	Memos
00:00:00	C1 Prepping the user/ setting up expectations			Motivating for why he did what he did	Talking about previous coding experiences? What led them to these questions, their motivation for showing this code. Previous experiences impact the code they built and are therefore showing??
00:00:05					
00:00:10					
00:00:15					
00:00:20					
00:00:25					
00:00:30		B1 Slides for background story			

Figure 3.2: An example of the coding performed by the researchers using an Excel spreadsheet. By coding using Excel spreadsheets, the researchers could also write memos as needed. Here the coder has written down a number of questions they have about the narrator's motivations for creating the video: *"Talking about previous coding experiences? What led them to these questions, their motivation for showing this code. Previous experiences impact the code they built and are therefore showing??"*

$$Percentage of Agreement = \frac{Number of Agreements}{Number of Agreements + Number of Disagreements} \times 100$$

Figure 3.3: The Percentage Agreement equation used to calculate inter-rater reliability in this study.

Concept Mapping

After a set of codes had been established, my collaborator and I experimented with categorizing the codes in a variety of ways to establish connections. For exploring the relationships between codes, a mapping activity was performed throughout the data analysis. Charmaz describes concept mapping as a method that helps the researcher "organize and conceptualize" their collected data [18]. Following the advice of Charmaz, we used this method to determine the relationships between codes and categories [17].

During the mapping activities, we took Post-it notes with the codes on them and arranged them on a board. We put them into groups spatially and lines were used to denote relationships. An example of this process can be seen in Figure 3.4. Throughout this activity, we tried to group codes in different ways to bring out assumptions about the codes not yet captured in the codebook definitions.

3.5 Phase 2: Interviews

Phase two of this research relied on interviews as a data source. This data was used to answer RQ2, "Why do developers create code screencasts?" and RQ3, "How do developers produce code screencasts?"

A total of 10 semi-structured interviews were conducted for this research. The length of the interviews ranged from 20 to 40 minutes. Skype and Google Hangouts were used in 9 out of 10 interviews, with the last interview taking place in person.

3.5.1 Interviewee Selection Process

The interviewees were discovered based on similar criteria used to determine the videos in the analysis. While authors from the set of analyzed videos were contacted, only two responded. A number of the screencasts in the sample were under corporate accounts, or accounts that had not been active for a long period of time; these authors were not contacted. Other video creators were contacted on the criteria that they had created a public video that contained the terms "Code walkthrough" or "Code Tour" in the video description. In total, 15 developers were contacted. Those who responded to email or social media requests for interviews were included in this study.

The participants in this study were all English speaking, males, with some education in Computer Science. All had programming experience outside of academia,

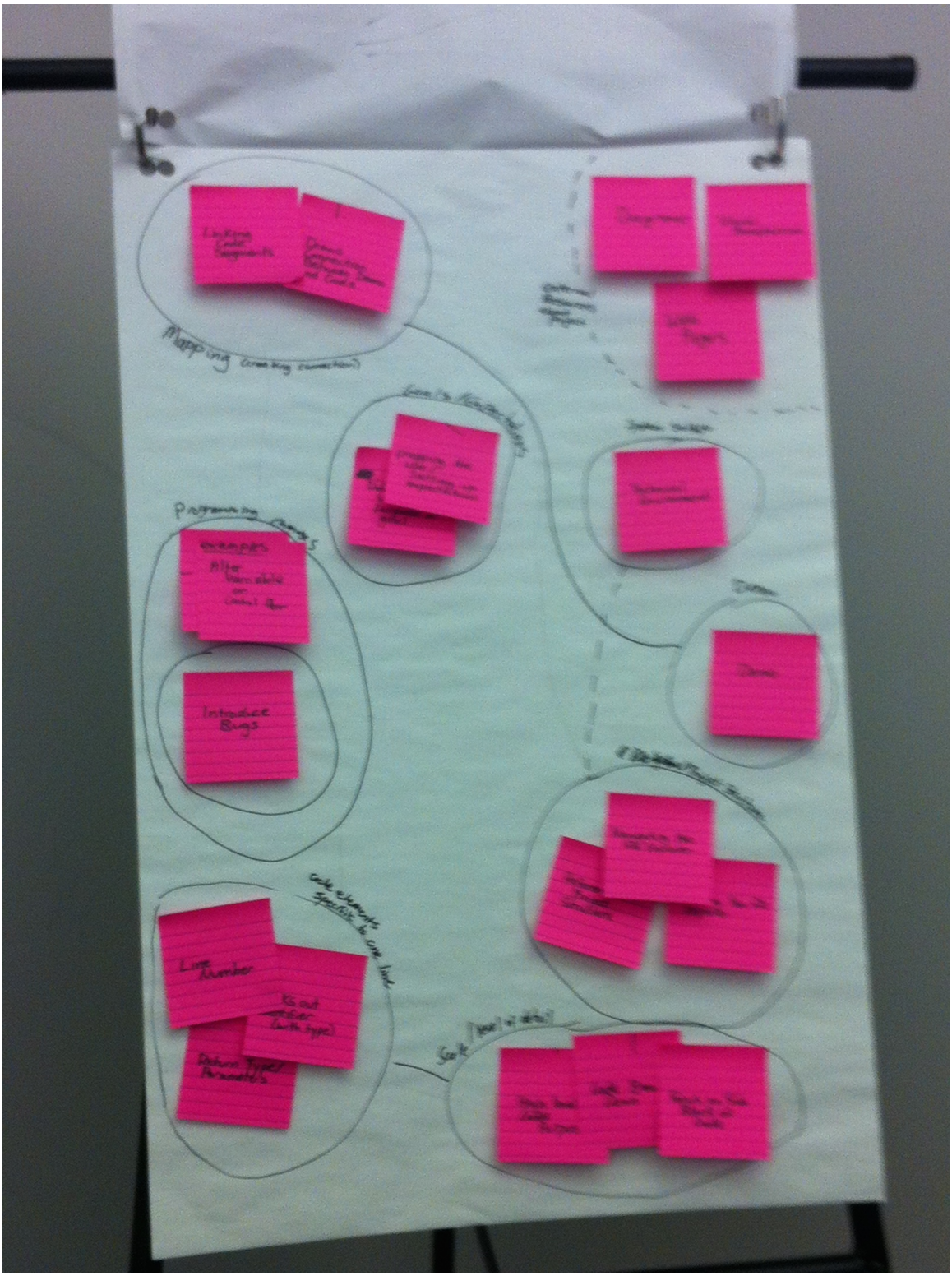


Figure 3.4: A photo of one of our concept mapping sessions.

though not all were active developers at the time of the interviews. Participants came from Europe, North America and Australia. Three of the participants were educators (formally or currently).

3.5.2 Semi-structured Interviews

The interviews were conducted in a semi-structured style. Semi-structured interviews are made up of, "a mixture of open-ended and specific questions, designed to elicit not only the information foreseen, but also unexpected types of information" [69].

Interviews began with an explanation of the research at hand and a verbal confirmation that the participants consented to having audio recorded during the session. Audio was recorded using an iPhone4, and was later transcribed for analysis. The questions used to guide these interviews can be found in Appendix B.

Participants were asked how they began making screencasts, and their production process. They were also asked about their background and opinion of screencast characteristics.

At the end of the interview, participants were asked if they had any further questions, which I answered to the best of my ability. Instructions were given to the participants to contact me by email, if they had any follow up questions or concerns.

3.5.3 Content Analysis

My collaborator and I transcribed the resulting interview audio files. This text was then analyzed using a content analysis approach. Based on the work of Krippendorff, content analysis is described as a "research technique for making replicable and valid inferences from data to their context" [48]. Using this method, researchers identify themes that run throughout multiple interviews [38].

We individually analyzed an initial set of four interviews, identifying categories. We went through looking for common ideas and experiences. We then presented our findings to each other. A set of categories was established, through discussion and iteration. The categories can be found in Section 4.2.2 of this thesis.

As the data was collected and analyzed, findings began to emerge. This chapter has outlined the methodology used in this thesis. Relying heavily on Chamaz's grounded theory methods, 20 screencasts from YouTube were analyzed and coded.

I then interviewed 10 developers with screencast creation experience. Data was collected in order to answer the four research questions concerning screencasts for developers.

Chapter 4

Findings

This chapter presents the findings of the techniques used by developers in screencasts. These are supported by observations from screencasts and interviews with developers. Chapter 5 brings together the findings in a theory which explores the techniques developers use to create screencasts.

4.1 Research Question 1: What Kinds of Program Knowledge Are Captured in Screencasts?

Developers create screencasts to convey and share technical information. Different underlying goals were found in the screencasts analyzed for this work. Developers were observed creating screencasts to share insights into how the code could be customized, to reflect on their development experiences with the program, and to demonstrate interesting implementations. It was also observed that developers created screencasts to share technical data structure and language-specific knowledge.

4.1.1 Sharing Customization Knowledge

Five screencasts in the sample showed how programs could be customized. The purpose of these screencasts was to explain the functionality of the program to the audience. This provided the audience with development details that could be used to repurpose the code.

Narrators were observed pointing out features and variables that could be changed to meet the audience's needs. For example in V11, the narrator showed how other developers could change the sound files used by the program.

4.1.2 Sharing Development Experiences

In 8 of the sample videos, narrators shared their development experiences by discussing the development process used to create the program. This included features that the narrator had not yet implemented or ways of improving performance. For example, narrators described where the audience could expand upon a program$_{V01,V02,V03,V7,V8,V11,V19,V20}$.

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

4.1.3 Sharing Implementation Approaches

Narrators use screencasts to share problem solving solutions with developers that may face similar situations. 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 narrators approach. Throughout the video, the narrator shares his inspiration and the challenges he faced.

In another example, V03 demonstrates a custom Minecraft launcher developed by the narrator. The narrator seemed proud of his program and wanted to share his approach with the Minecraft community. He explained the challenges he faced creating the user interface.

4.1.4 Demonstrating the Application of Design Patterns

Screencasts are used to demonstrate how design patterns are used for implementing solutions. In V02, the narrator states that he made the video to demonstrate how to

implement responsive design principles in Lua. His goal was also to share language-specific knowledge through a screencast. The video demonstrates design principles that rely on an understanding of the programming language, Lua. A video approach allows the narrator to demonstrate the implementation of these design patterns while executing a finished, real world solution.

4.1.5 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 the language itself, as opposed to implemented programs or patterns.

This technical knowledge helps the audience utilize and understand the programming language. For example, V10 provided a walkthrough of the Ruby programming language. In this video, the narrators walk through 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 show a deep level of understanding of the subject matter.

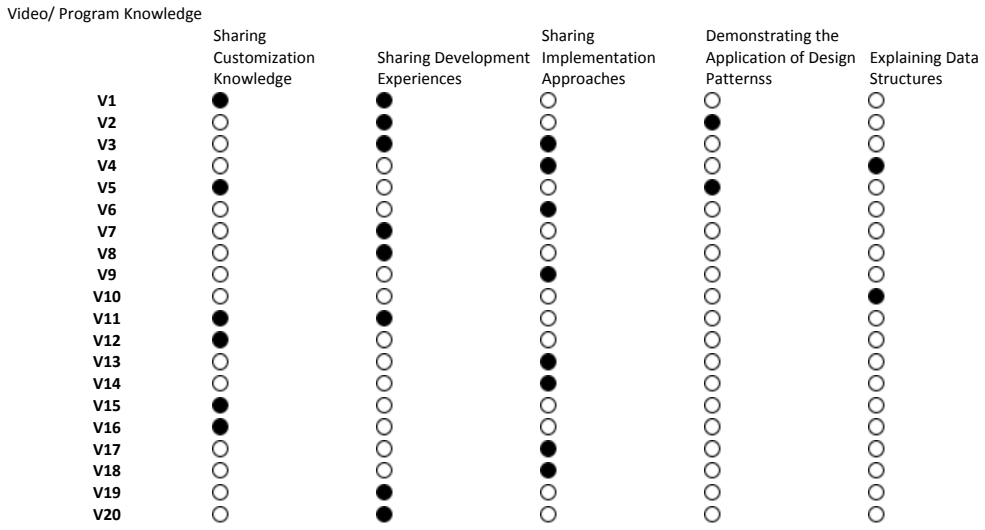


Figure 4.1: This chart shows the program knowledge goals in relation to the videos in our sample. A black dot indicates that the video demonstrated the program knowledge goal.

From analyzing the set of screencasts, five goals were distilled that these screencasts aimed to fulfill. Figure 4.1.5 shows an overview of these findings. This research question provides insight into the types of screencasts that developers create to share technical knowledge.

4.2 Research Question 2: What Techniques do Developers Use to Document Code in Screencasts?

Screencasts were analyzed in this study to understand the techniques developers use to present code through video. In this study 20 screencasts hosted on YouTube were analyzed to distill these results. The codes developed from analyzing the screencasts are defined in Appendix A. Both these codes and the following themes were refined using diagramming activities. The codes and themes are discussed in further detail in Chapter 5.

4.2.1 Codes

Chapter 3 described how a codebook was created for the screencast analysis phase. The codes in the codebook were developed over many sessions — Figure 4.2 shows my notes after an initial coding session. Originally, each code had a letter-number pair which was used to make coding the screencasts easier. For example, "Defining the video purpose or goal" was originally given the code of A1. In the final version, these letter-number pairs were removed and all codes had descriptive names.

After iterating on the codes and definitions, a diagramming exercise was used to explore relationships between the codes. Figure 4.3 shows the results of an early diagramming process. The image shows codes that occur together, or rely on each other to share information. For example, the code "Draws connection between demo and code" is linked to both the "Demo" and "Picks identifier with type."

4.2.2 Themes

After the codes were refined, they were grouped into themes. Figure 4.4 shows one instance where codes were mapped into themes, again using a diagramming activity. In this way, we built off previous analysis to refine the results.

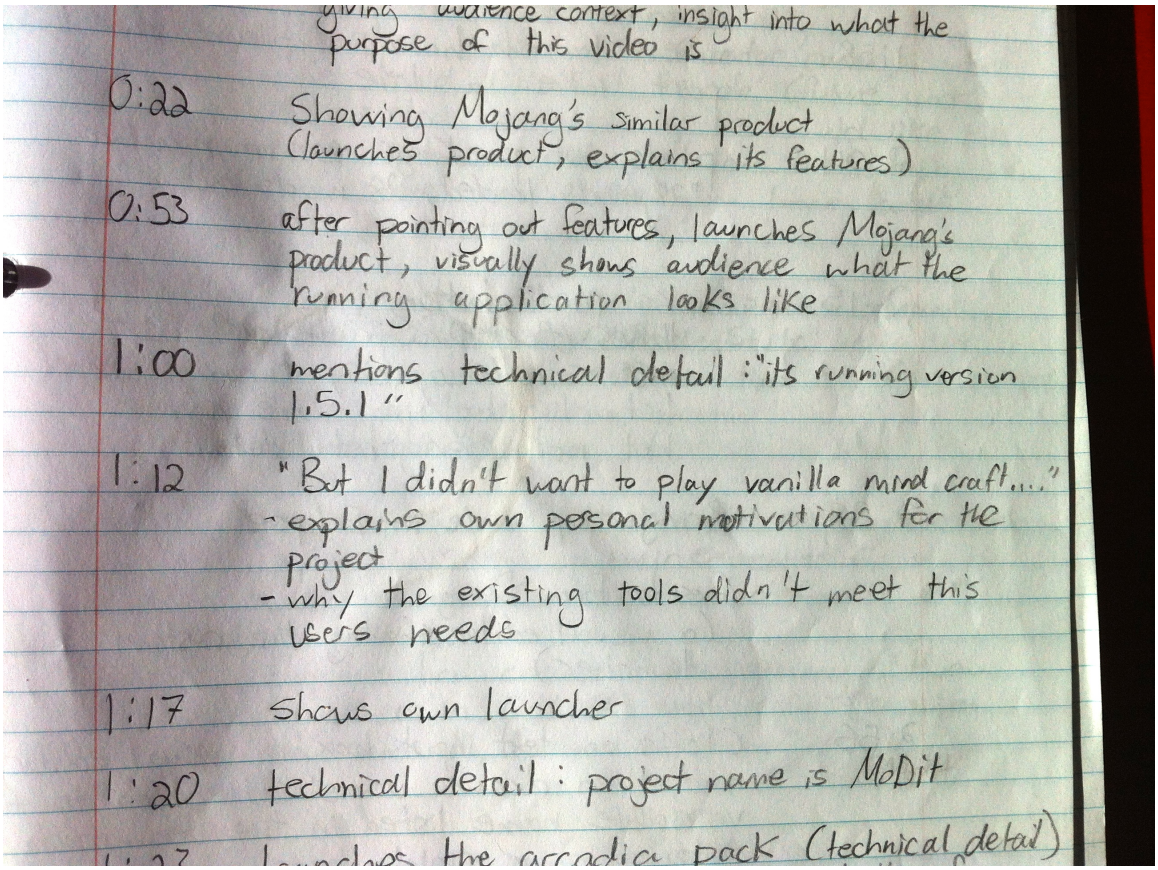


Figure 4.2: A sample from an initial coding session. The entry at 1:12 shows a quote from the screencast where the author describes their motivation for creating this application.

The final themes and their corresponding codes can be seen in Chapter 5. They are presented here to show the relationship between the coding and categorization phases of this study.

4.3 Research Question 3: Why do Developers Create Code Screencasts?

The third research question focused on why developers create screencasts. From the interviews, it was found that developers created screencasts for a number of different reasons. Creating screencasts contributed to a developer's online identity and credibility. It also allowed them to promote themselves and their work. Interviewees spoke of using screencasts as a way to teach themselves new material. YouTube was

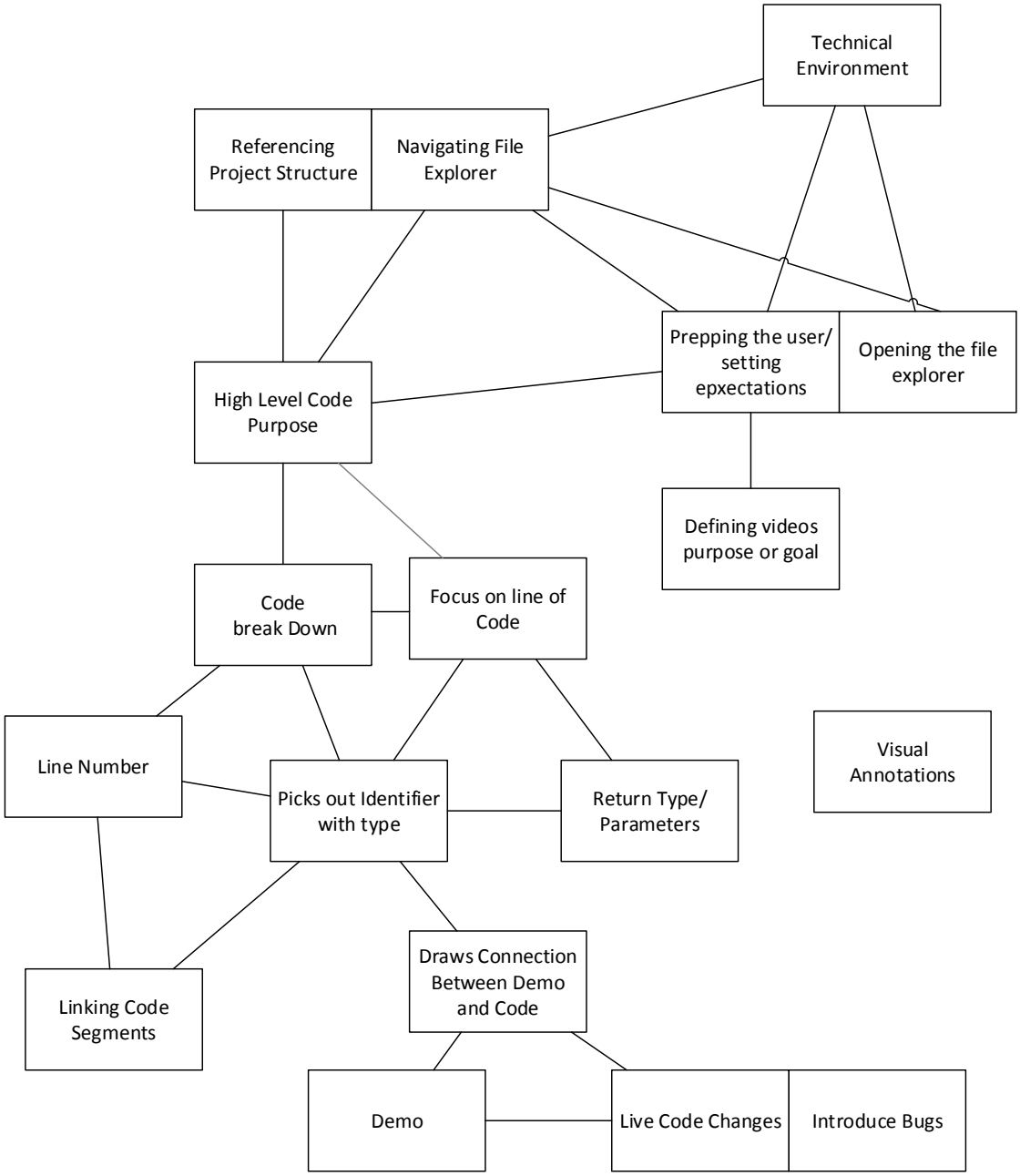


Figure 4.3: The resulting map after codes where mapped for the first time. Each code is shown in a box. Codes that shared a strong relationship are touching. Other relationships are denoted through solid lines.

found beneficial by some interviewees because it allowed them to reach a greater audience.

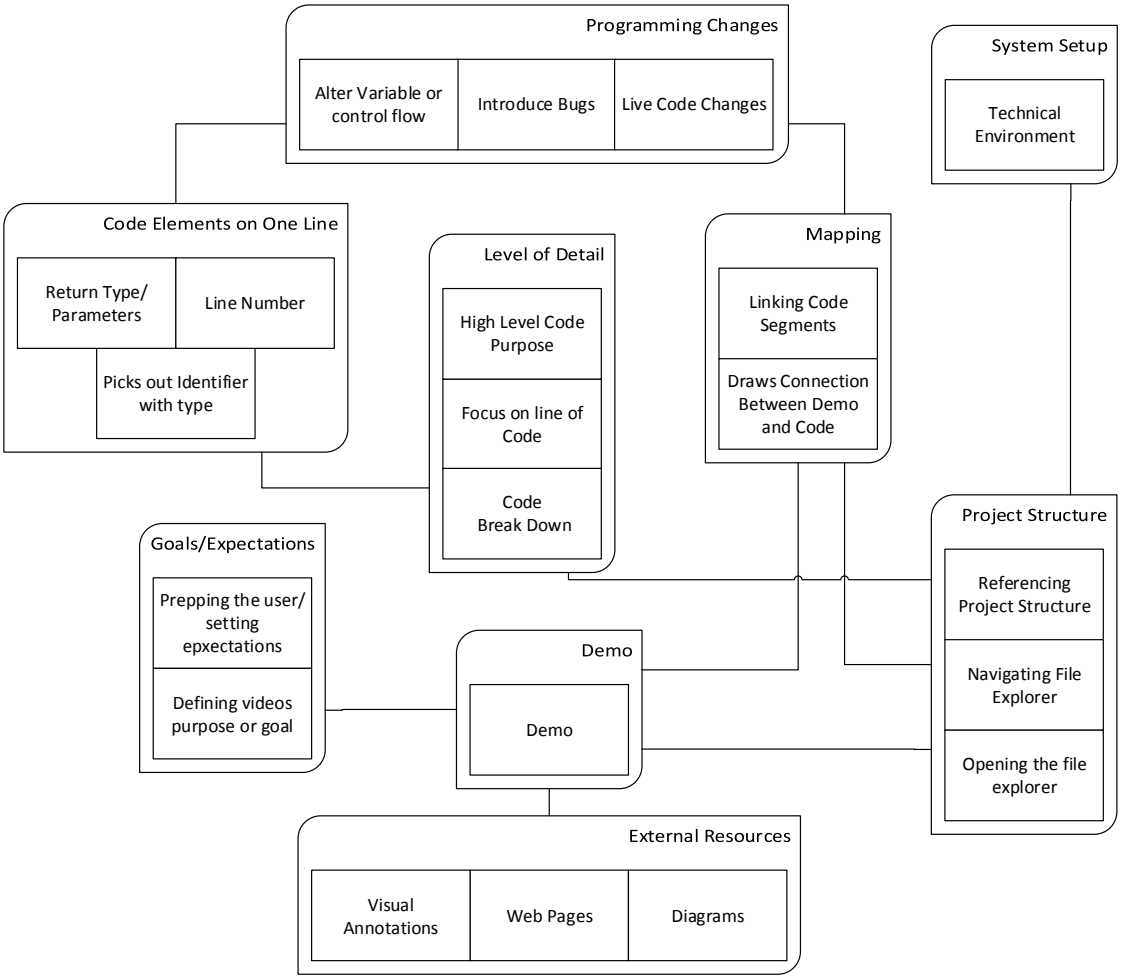


Figure 4.4: A resulting map after coding multiple times. Andreas and I used an adjacency coding technique to map different relationships between the codes. As per Charmaz, we tried multiple organizations to distinguish underlying assumptions about each code [17].

4.3.1 To Build an Online Identity

By crafting an online identity, Begel et al. found that developers use social media outlets to promote themselves and build a presence [8]. As previously mentioned in the literature, developers also use social media to evaluate other programmers and projects [27]. Among the developers we interviewed, a common theme emerged of using YouTube to contribute to their existing online identity.

For example, P07 reported that making videos made him more credible to potential employers. P05 reported that screencasts provided him with a type of authority on the subject matter:

> "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."

For P06, YouTube videos not only let him establish his work through content, but also gave him a way to create relationships with his audience:

> "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 much more natural relationship with your audience."

It has been shown that developers choose to share content that will enhance their online image [70]. A similar trend was found in two of the interviewees in our study. These two developers shared how they had created screencasts, but not released them. They cited not doing so because of issues with the quality of the screencasts$_{P08}$, or because they felt the topic would attract negative attention$_{P07}$. As P07 explained:

> "So I've done that a couple times, where I've recorded the video, I felt really good, [but] I never released it. It was four hours and I was ready to go and I thought this sucks, or I don't like it because I know all the trolls on Reddit will have issue with it... So I have to be very careful about what I release out there."

In this way, the developers in our study were actively curating their online image through what they chose to share (or not share) via YouTube.

4.3.2 To Promote Themselves

During the interviews, developers spoke of using YouTube to promote themselves for personal or financial gain$_{P03,P10,P05,P04}$. In the case of P10, he had begun by posting his videos on Udemy as a way to make money. YouTube gave him a way to reach a greater audience, as opposed to a closed, paid model. When he posted his videos on YouTube, he explained:

> "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... I have a lot of blog followers now and I have a couple thousand YouTube subscribers [up] from thirty."

Four other interviewees used YouTube to broadcast their companies or products $_{P04,P06,P07,P10}$. In this way, YouTube was used as a promotion tool.

4.3.3 As a Learning Exercise

Interviewees spoke of how creating videos forced them to better understand the topic at hand$_{P02,P03,P07}$. P01 described how creating screencasts forced him to sit down and improve his skills every week: he created a video once a week to stay up to date on developments in the Blender community. By creating screencasts, interviewees also described feeling more confident in their understanding of the material$_{P07,P03}$. P07 explained:

> "I mean in creating it you have to know something well enough to articulate it. So I'll record myself ... and I'll re-watch it and I'll say dude I don't know what I'm talking about! So I'm going to go research again ... and people will go like "yeah you must have been doing this for years." And really it took me an hour to learn that." $_{P07}$

By doing this, developers used screencasts as a way to build upon existing knowledge.

4.3.4 To Give Back

A common finding was that developers made videos to address what they wish they had known when they started programming$_{P01,P03,P07,P08}$. Developers described this phenomenon in terms of the amount of time they could have saved$_{P03,P07}$. This has been noted in other studies where developers feel that they need to record their experiences to keep others from going through the same struggles [71].

Three of the interviewees were teachers and two of them spoke of how they started creating videos to help their classes. P03 talked about how his screencasts impacted his class:

> "They said it was really helpful ... and I actually saw it reflect in the way they were writing their code, so these are the things I wish I had when I went through the same thing, and that's my guide in how I teach."

By creating these screencasts, developers reflected on their personal experiences and tried to help other programmers. Developers do this for altruistic means with

the aim of helping people in their community [50]. From this study, interviewees discussed sharing knowledge by codifying their experiences through screencasts.

4.3.5 As an Alternative to Blogging

Four interviewees discussed how they felt that creating a screencast was a better use of their time than blogging$_{P01,P07,P08,P09}$. P08 stated his preference:

> "I know that like 10 or 15 minutes is still going to be faster than actually going in and editing, at least for me."$_{P08}$

As video creators, they felt that they knew the information well enough that it was easier for them to record and capture their screen, than to go through the writing process. P03, P07 and P08 described that, as viewers, they also preferred to learn from video over static 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."

However, not all interviewees felt this way. P05 talked about how he created videos for a year before deciding that it was not worth the effort put into creating them and that he didn't find them useful for learning. He said:

> "I can read much faster than most people can speak. I would watch tool demos— how do I make the tool do this thing. I wouldn't watch it for factual content because I can read."

Video is not the ideal learning medium for everyone, but these interviews show that there is a community who uses this method for acquiring and sharing knowledge.

Through the interviews, participants described creating code screencasts for a number of reasons. It helped them establish their credibility and identity online. It provided them an alternative to blogging by sharing information through video, and it allowed them to promote their projects and skills. YouTube screencasts allowed our participants to build relationships with an audience. Interviewees reported screencasts as having positive effects on their career or developer identity.

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

Table 4.1: A table of the motivations found from interviewing screencast creators.

4.4 Research Question 4: How Do Developers Produce Code Screencasts?

In the interviews, participants were asked about their creation processes. Here, the findings on the screencast creation process are presented in three common stages: preparation, recording and post-production.

4.4.1 Preparing the Screencast

Before recording, the interviewees stressed the importance of organizing their thoughts and sources. Almost all of the interviewees, with the exception of P03, spoke of using an outline to note important points they wished to address in their videos:

> "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" $_{P06}$

Here, P06 used an outline as an initial planning tool. He did not actually use the outline while recording the video. It served him as an organizational tool. Interviewees also spoke of also writing the code they were going to use in the video, or gathering images and diagrams$_{P01,P07,P08}$. The planning stage consisted of participants identifying key information, and gathering materials for recording. Interviewees reported using outlines to organize their thoughts and highlight important pieces of information.

Interviewee descriptions of the planning processes resemble the findings of educational research on screencasting [79]. Previous work describes the planning stage as forcing the creator to focus their ideas and goals [57]. By planning ahead, the

literature suggests that screencast creators can tailor their video to meet the needs of their audience [61].

4.4.2 Recording the Screencast

When recording video, the participants used a variety of screen capture tools. Tools used for recording included Quick Time, Camtasia and Screenflow, though no one tool dominated in the sample. While recording, the interviewees stressed the importance of breaking clips into short segments$_{P03,P04,P05,P06}$. This seemed to be for two reasons: it limited the possibility of mistakes, which would require to a retake$_{P05,P08}$; and, it forced them to articulate their ideas clearly and quickly$_{P02,P03,P04,P06,P08}$.

When interviewees did encounter mistakes during recording, some noted interesting coping strategies. In this situation, interviewees explained that instead of turning off the camera, they simply debugged live$_{P01,P08,P10}$. As P01 explained:

> "I just live type and if it doesn't work it's really awkward, but we fix it then because they learn debugging." $_{P01}$

Thus, programming mistakes are used as teachable moments that interviewees use to show the viewer how to get around problems that they also might experience while following the screencast. This is a unique situation to programming, where by capturing the execution of a program live, one can turn errors into teachable moments. In this way, developers can give their audience real insights into the functionality of the program.

4.4.3 Post-production

After recording, interviewees took the time to edit their screencast. Out of ten people, seven told me that they edited their work$_{P01,P04,P05,P06,P07,P09P10}$. This ranged from adding effects, to reworking the audio. For example, P01 put hours of effort into editing his work. He said:

> "I'll come over to the audio and spend 2 hours just combing through getting rid of clicks" $_{P01}$

Of those who edited their work, interviewees stressed the importance of high production quality$_{P04,P05,P07,P10}$. For example, P10 shared the story of how a man with a hearing disability was able to understand P10's screencasts because of the audio quality. In an interview P10 said:

> "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." $_{P10}$

Alternatively though, a group of participants expressed that they did not do any editing and simply watched their videos for clarity before putting them on YouTube$_{P08,P02,P03}$. One example was P08, who said:

> "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." $_{P08}$

The majority of people I interviewed reported doing some form of editing as a quality measure. High production quality was important to these individuals$_{P04,P05,P07,P10}$. Others felt that editing was not necessary.

Those who were comfortable with their editing tools demonstrated a number of different techniques they used to edit out errors. P07 explained how he accepted the fact that he would make mistakes and how he had strategies to compensate. This included using YouTube comments or annotations on the screen recording. P10 described how he adapted to his software support to identify errors quicker in the post-production phase:

> "I'll snap my fingers and stop talking, so it will peak the sound wave, so when I am editing I can see where I snap my fingers and I'll know that I made a mistake there." $_{P10}$

Others mentioned recording in short segments to reduce the chance of making an error$_{P06}$, and hiding the clock on their desktop to make editing clips together easier$_{P04}$.

A common theme in the transcripts was participants expressing frustration with the tools they used to create videos$_{P01,P04,P06,P07,P08,P09}$. Sources of frustration included editing and recording software, upload times, quality, as well as adjusting to screen resolutions. Interviewees reported that they spent the most amount of time in post-production improving the quality of their work$_{P01,P04,P07}$. Though many different tools were used in the sample, none of them were found to be ideal by our participants.

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

Table 4.2: A table showing how interviewees described producing screencasts

In this chapter, the findings from our analysis of the data were presented. From analyzing YouTube screencasts and interviews, the four research questions were answered. From these questions, a number of themes emerged, addressing how developers present code in video. These themes are outlined and drawn together in the next chapter, Chapter 5.

Chapter 5

Theory

> "The result of a constructivist grounded theory study is more seldom presented as a theory than as a story or a narrative, including categories, told by the researcher with a focus on understanding of social processes" Lillemor Hallberg, 2006 [40].

After applying grounded theory, one is expected to produce a theory encapsulating the observations and insights they have gleaned from analyzing the data. As Charmaz describes, through grounded theory we theorize how meaning, actions and social structures are constructed [20].

As a result of my constructivist approach, this thesis does not present a traditional, positivist theory— variables and processes are not an output. On the contrary, this theory presents groupings of analytic statements, supported by description and illustration from the data. What is presented in the following section is a description of categories based on an interpretation of screencasts and interviews. This interpretation explores the experiences of developers who share screencasts via YouTube.

In this theory, screencasts and interviews are defined as data. The data was analyzed using a qualitative approach. Themes were interpreted from the data and seen as indicators of shared experiences. This work seeks to situate screencasts in the social ecosystem that developers use to share knowledge. Through this, the thesis aim to provide context for screencasts, as well as insight into how and why developers create them.

5.1 Context

The next subsections provide an overview of the context of the findings. By drawing together observations from the findings the time, place, culture, and situation that the theory occurs in is described.

5.1.1 Time

YouTube is a popular online video sharing platform that contains a social media features that allow people to communicate and share ideas. In temporal terms, this theory applies to one of the most popular, contemporary social video sharing sites.

The screencasts we studied were created within the last six years. The technology for creating videos has become drastically more accessible in recent years. Many computers and phones now come with entry-level hardware and software for video recording.

Since its launch eight years ago, YouTube has become extremely popular. Over 100 hours of film are uploaded to the site every minute on a range of topics [87]. For example the interviewees mentioned using YouTube not only to post their own videos, but also to find entertainment and educational material$_{P01,P02,P08}$.

5.1.2 Place

When posting on YouTube, one must choose to make their video public or private. For this study, public videos were analyzed. It is assumed that the creator wanted others to be able to view their work.

Whether consciously or not, the interviewees held assumptions about their intended audience. Sometimes these assumptions were explicit. For example, P02 and P03 described making videos for a classroom setting, assuming that their audience had a very basic understanding of programming concepts. In that case, they had a clear understanding of the background and needs of their audience.

Other times, interviewees described having no particular audience in mind, but that their screencasts focused on topics based on their programming experiences$_{P07,P08}$. In this case, the video creator still has assumptions about the audience's skill level and background.

These assumptions can cause problems for the screencast creator. For example, P01 began creating screencasts videos for "beginners and beyond." When he made an

advanced series, he found that audience members struggled with the more advanced topics.

> "But I'll get questions from people who are clearly new to it but they have seen the rest of my channel and they have seen the easier tutorials I have done, where I say every click, 'now we press the left mouse button, now we press these keys.' And they assume I am going to do that for the harder things too." $_{P01}$

When he began creating videos for a higher skill level, he received questions from audience members who didn't have the skills needed to follow the video. They weren't the audience he had in mind.

The openness of social media means that anyone can access public content. Content creators need to be aware of this and communicate their intent when posting public videos on YouTube. For developers, the platform they use to share their knowledge impacts who can access their work. As the example of P01 demonstrates, when content creator and audience expectations do not align, this creates friction.

5.1.3 Culture

Social media has been described as contributing to the rise of a participatory culture [75]. By participating in social media, one subscribes to the cultural values and social norms of the platform. This shapes how one participates in this space.

According to Dabbish et al., by sharing their work, developers are contributing to a larger social community [27]. Through a participatory culture, people are encouraged to share their ideas and believe that their contributions are important [86]. Users have been found care about how they are perceived in this space and act in accordance with what they believe to be the cultural norms [27].

Singer et al. put forth the idea that through social media, developers have formed communities of practice [70]. These communities are built around software, tools and shared interests [70]. As Wenger describes, there are three ways for a person to show their competence in a community of practice: they need to understand the community well enough to contribute to it; they need to interact with other members and demonstrate their understanding of the community; and they need to show that they understand the "language, routines, sensibilities artifacts, tools, stories, styles, etc." of the community [86]. By demonstrating these, a person can establish that they belong to the community. Programming communities influence the experiences

developers share in screencasts and how they do so. For example as previously mentioned in Chapter 4, interviewee P07 explained how he has created screencasts, but never released them because of the negative attention he thought the topics would attract online. P07 later stated:

> "I realized that if I'm gonna explode on the Javascript scene, I have to work really hard to stay up there, to stay relevant. " $_{P07}$

In this way, P07 had taken and internalized social cues from members of the Javascript community. His previous interactions with the community guided what and how he contributed. He wanted to make videos that fit what he perceived to be the social norms and views of the community. Based on his previous experiences, P07 had formed ideas of the types of content that would attract negative attention.

The interviewee demonstrated that he cares about this identity within the community. He wants to be known as an expert and identifies how he needs to act in order to achieve this. His perception of the expectations and behaviors of the Javascript community impacted his choices and shaped his participation.

Developers who share screencasts via YouTube are not only part of a participatory culture that urges them to contribute their knowledge, but also part of communities of practice, with social norms they must navigate. Wenger states that these communities influence the language, behavior and actions taken by members of the community [86].

5.1.4 Situation

This work explores the experiences of people who have developed software and chosen to share their experiences through a screencast hosted on YouTube. They may or may not have written the code in the screencast, but they have some knowledge of the program. It is assumed that, through screencasting, the screencast creator will share their experiences and understanding of the program. This does not mean that their understanding of the program is correct. One assumes that sharing these experiences imparts knowledge on the audience in the form of explicit or tacit declarations.

A number of reasons may motivate the screencast creator to share their experiences through YouTube, such as the desire to establish an online identity and wanting to give back to their community. In this thesis, screencasts are observed as an artifact of the developer experience and support this study's analysis of these artifacts with insights from screencast creators. The following sections blends an analysis of real

world screencasts with interviewee data to describe the techniques used in screencasts hosted on YouTube.

The techniques described in this section apply to situations where one uses screencasts to share their knowledge about a computer program, specifically in the YouTube environment. These make up a theory that describes the techniques developers use in these videos.

From the analysis, it was found that developers use a variety of techniques to document and describe their code. These techniques are used in conjunction with each other to allow the narrator to share their knowledge, while interacting with the program.

The narrator of a screencast is understood to be a developer who has knowledge of the program at hand and can refer to someone who has actively developed or maintained the program. Situations might arise where one has detailed knowledge of the program without this development experience.

Ultimately, the screencast under analysis explains the functionality of a completed, non-trivial program or code snippet. In creating the screencast, the narrator formalizes their knowledge of the screencast—they walk through the screencast to build an overarching explanation of the program.

5.2 The Theory

To begin a screencast, the narrator starts by *defining the video's goal* for the audience. They communicate their intent and explain what will (and perhaps will not be) covered in the screencast. Through this, they communicate the subject and purpose of the screencast.

As the narrator moves through the program, they make *references to the code*: its architecture, how the files are arranged, and how data flows. They focus on specific functions, explaining purpose and parameters. They *link* parts of the code together by moving from one section to the next and explaining relationships.

Eventually, the narrator decides to *execute the program.* They do this to show the audience the final output. They draw attention to features seen in the demonstration and link them back to *references in the code.* Through the act of linking, they build up the audience's mental model of the program.

But maybe all the answers aren't found in the code? The narrator may provide the audience with knowledge of related documentation and where to find it. These are *references to external resources* for the audience members to refer to.

It is possible that the program crashes during the demonstration. The narrator begins live debugging to fix the issue. They *manipulate the code* to show the audience where variables need to be changed and relaunch the program to demonstrate the correct execution.

The narrator may use many different techniques to the showcase the program. From the analysis, seven themes emerged.

5.3 Themes

The following subsections explain the themes of this theory. The themes are made up codes. *Themes* are presented in italics and as subsections in the following pages. The **codes** of each category are presented in bold. From the codes and themes, this work described how developers use a number of different techniques to document code in screencasts.

5.3.1 Goal Setting

Goal Definition requires the narrator to explain their intent for the screencast. This includes who the screencast is for, and what topics the screencast will cover. The narrator sets the stage for the audience by communicating what to expect in the video. In a screencast, these messages are typically communicated verbally, but they can also be inferred from video descriptions or images.

This technique allows the audience to decide if the screencast suits their needs. The theme *Goal Setting* was used to describe this phenomenon. From the interviews, it was found that narrators used this technique not only to help audience members, but also to help themselves. They explained how defining the purpose of the video helped them focus their videos.

Every screencast began with a statement of the goal of the video. The code **verbally defining the video's purpose** was used to capture this. From RQ1 it was determined that narrators make videos to achieve a number of different goals.

Narrators set up audience expectations by providing **an explanation of the limitations (i.e., the scope) of the video and its intended audience**. For

example, the narrator of V01 remarked how the project relied on a specific server, but that instructions on how to set up the server would not be covered in the video. This allows the audience to determine if the screencast suits their needs. The interviewed developers expressed that focusing the goal or point of the video helped them focus their creation process$_{P02,P05,P06,P07,P08,P09,P10}$. As P09 explained:

> "They can just refer to that little video that covers that topic... I like splitting up the video into bite size pieces that can just cover what they are gonna cover, so [that] you can just navigate to the video that covers that bit of information." $_{P09}$

The interviewees described how their audience members became frustrated when the purpose of a screencast was unfocused or not clearly communicated. P05 noted that there is no way for the audience to skip to the relevant part of a video on YouTube. Because of this, interviewees tried to help their audience find knowledge by making their videos shorter:

> "I'm trying to refine it to literally have videos as short as possible. If they can be two minutes perfect, so some of my videos are literally one minute to the most" $_{P10}$

Research on software engineering documentation has found that developers avoid documentation that is, "complex and time consuming" [49]. They are drawn to artifacts that are easy to understand and straight to the point [49]. Our screencast creators comments echo this and show that they are aware of how it impacts their audience.

Audience members need to know if a screencast will helpful to them, and may not have time to watch the entire video. Audiences will instead watch parts of multiple resources quickly, before deciding to focus on one that may suit their needs [11]. Similarly, developers need to quickly judge if a resource is relevant to the task at hand.

Defining the screencast purpose is similar to ideas from the literature about an, "overview strategy" which provides, "the necessary background information that learners need in order to understand the context and/or the purpose of the screencasting topic" [79]. Outlining what will be covered in a screencast sets up the audience's expectations. From the interviewee's perspective, having a focused video is one way in which screencast creators can tailor their approach to help their audience.

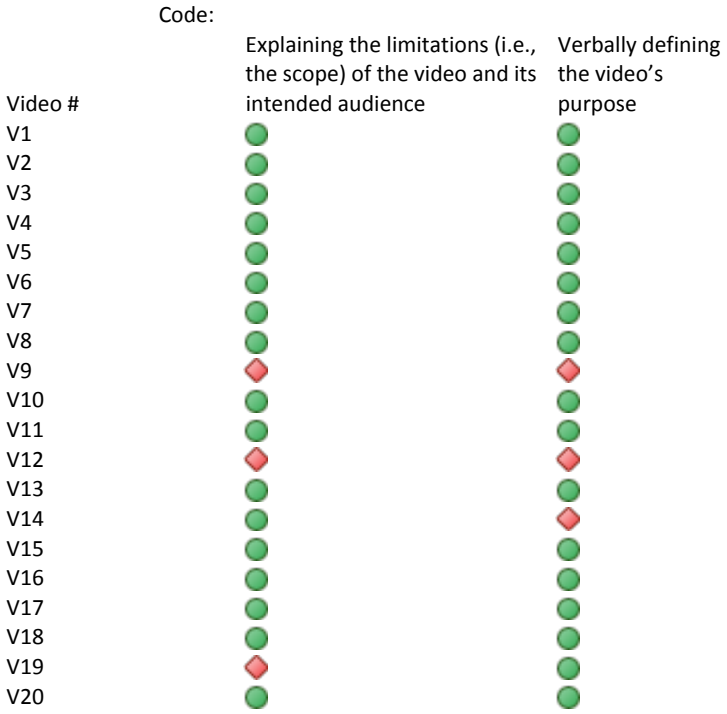


Figure 5.1: The results from coding the set of videos for the goal setting theme.

5.3.2 Referencing Different Levels of Detail

Narrators were observed describing features and components using various **levels of detail**. For example, features that ran across many files might be summarized by the narrator using **high-level** descriptions. In V02, the 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. This work took a pragmatic approach and used the term **medium-level** for descriptions of blocks of code, such as a logical sequence, that spanned multiple lines. Narrators used this level of detail to summarize sections of code quickly, 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]."

This was contrasted by narrators describing **low-level**, or in-line, language-specific information. Frequently this meant reading the code aloud, verbatim. For example, the code in V01:

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

was read aloud:

> "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 also referenced specific **element identifiers**, such variable names or types, and explain their functionality to the audience. They also picked out **return types, parameters and line numbers**. For example, the narrator of V04 picked out the variables "Name", "User" and "Password", to explain how to set up the connection to a local database. These techniques were used to explain software features and functionality.

Figure 5.2: The results from coding the set of videos for the referencing different levels of detail theme.

5.3.3 Browsing the Technical Environment

Narrators also used screencasts to explain the **structure of a program**$_{V1,V2,V4,V6,V7,V10,V13,V14,V16,V17,V18,V20}$. For example, the narrator of V04 provided a general overview the setting.py file's purpose:

> "This top-level settings file is just the top-level routing so its saying anything that isn't oautho2 or admin send it off to media urls file and it will handle that." $_{V04}$

For the following lines of code:

```
local loadingView = LoadingView:new(self)
self.loadingView = loadingView
```

This section of code acted as beacon to the developer about where a particular feature was instantiated.

Commonly, the **file explorer** was used to describe where code fit within the program's structure. Narrators visually outlined the structure of their programs and provided audio descriptions. These techniques provide context to the audience, which Sugar et al. suggests establishes links between code and features of the program [79].

By making references to the code, narrators choose where to focus the audience's attention. The narrator chooses part of the code as examples of "beacons" that Storey describes as "recognizable, familiar features in the code that act as cues to the presence of certain structures" [73]. Brooks writes that in order to understand a program, the audience must be able, "to create mappings from problem to code" [12]. To do this, previous work shows how narrators can use indicators in the text to navigate the program and identify features [74].

The idea of beacons is reflected similarly in the work of Pennington, who describes how developers build up a structural, mental, model of a program. This model relies on knowledge taken from the structure of text, and the "program plan" [84]. By understanding the relationships within a program, one develops their mental model of it [65]. In the example above, the narrator's program plan maps code to a feature– they create connections between the two.

References to the code are used in screencasts to focus the audience's attention. They are a way for the narrator to highlight parts of the program they feel are important and explain the program's structure.

Figure 5.3: The results from coding the set of videos for the browsing the technical environment theme.

5.3.4 Demonstrations to Showcase Execution

The theme of referring to the executable came from our screencast observations. Narrators executed the program to demonstrate the output to the audience. Sometimes the narrator would run the program multiple times to show the audience how different inputs, or changes to the code, impacted the program. The majority of screencasts in our sample **executed a program to demonstrate features to the audience** $_{V1,V2,V3,V5,V6,V9,V13,V16,V17,V18,V19,V20}$.

Narrators used demonstrations as an opportunity to explain software functionality $_{V01,V02,V13,V18,V19,V20}$. Demonstrations were also used to explain specific use cases and data flows within a program$_{V01,V02,V16,V20}$. Some screencasts with a graphical component made use of demonstrations to explain how a program manipulated user input$_{V05,V06,V13}$. In these ways, 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.

This technique allows the audience to analyze, "the properties of a running software system" [9]. Through screencasts, narrators show both the program output and the code. With this information, the audience creates reference points between the two.

By running a program, the developer gains insights into details of software behavior [23]. In this study, narrators used this technique to communicate information about their code to the audience. Video provides alternative insights into the coding process over static text. Having source code available, in conjunction, provides the audience with a working example to manipulate.

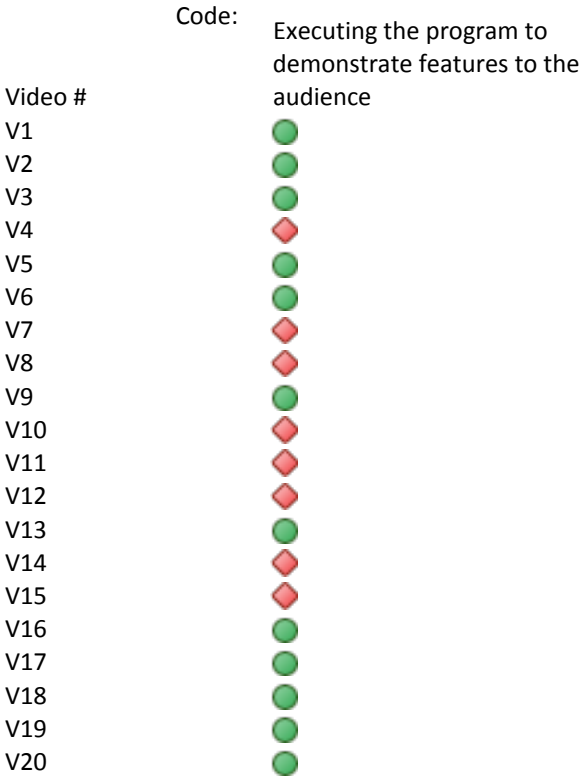


Figure 5.4: The results from coding the set of videos for the demonstrations theme.

5.3.5 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. It was observed that *live code changes*$_{V2,V3,V5,V6,V7,V8,V10,V11,V13,V14,V19,V20}$ were performed for two main reasons. One was to **change**

a program's control flow or variables$_{V2,V13,V19,V20}$, and the other was to introduce the audience to examples of increasing difficulty$_{V13}$.

Narrators also **introduced bugs** into the program, either intentionally or intentionally $_{V2,V3,V5,V13,V15,V20}$. 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 situation was turned into a live debugging exercise. Live editing also offers the audience a glimpse into the narrator's coding style, but it can also be used to manipulate the output of a program.

Because these videos present code, there is the possibility that bugs will appear unexpectedly. As mentioned, interviewees explained that instead of turning off the camera, they simply debugged live$_{P01,P08,P10}$. Thus, programming mistakes are used as teachable moments that the audience might experience while following the screencast. Live editing lets the narrator expand the audience's understanding of the program.

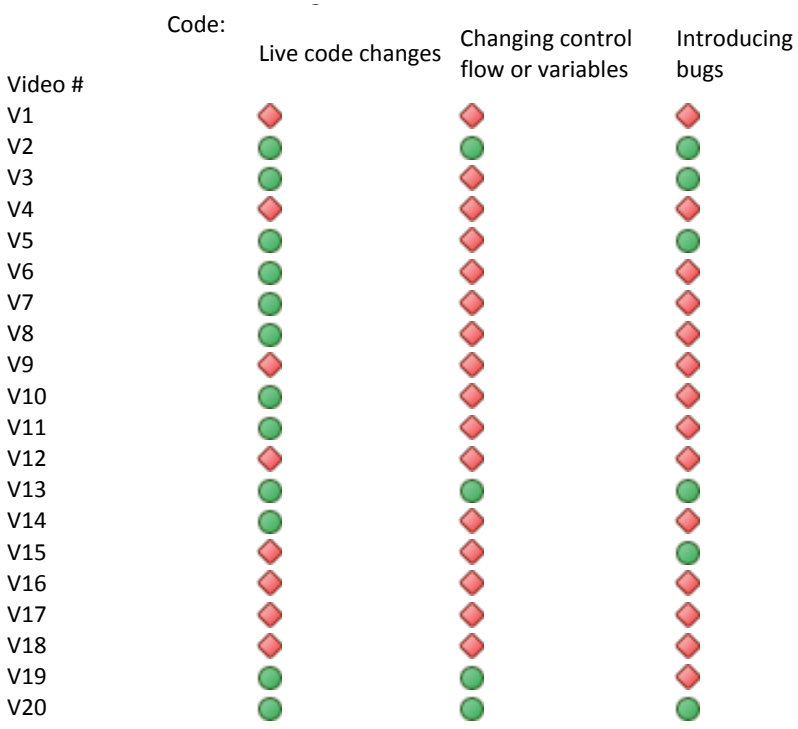


Figure 5.5: The results from coding the set of videos for the live editing theme.

5.3.6 Provisioning of Additional Resources

Not all information about a program is stored in its code. Developers frequently use other sources of information to support their work. This includes design documents, APIs, libraries and Websites. We observed developers make use of these resources in our analysis.

Screencasts exist as one type of knowledge documentation accessible to software developers via the Internet. Narrators frequently referenced other forms of documentation and external resources. This included **Webpages**, where audience members could find additional documentation, and **diagrams**$_{V16,V17,V18}$.

To correct mistakes, narrators were also observed using the **visual annotation** boxes provided by YouTube to share links or comments$_{V01,V03,V06,V10,V11}$. These techniques let the narrator provide the audience with text and video resources that supplement each other.

Source code was also frequently made publicly available and referenced in the videos$_{V1,V2,V4,V6,V9,V10,V13,V18,V19,V20}$. As P02 explained, he felt that source code was beneficial to the audience:

> "..but if you already understand the syntax and you're really just trying to learn how to use that technology, it's a lot slower to go type that stuff in than it is to download a repository from GitHub and get the source code, dig into it, and sort of watch along with the person and the video. That's kind of how I structure my videos, as almost all have a source code component that goes with it."$_{P02}$

This quote from P02 distinguishes between learning the fundamentals of a programming language and learning how to use it.

Interviewees reported providing source code for audience members to expand and build upon. An exception to this was V03, where the narrator explained how he specifically did not make his source code available. Interviewees stressed the importance of providing source code to the audience$_{P03,P04,P08,P09}$. This finding reflected what we had observed in the screencasts, where seven screencasts provided links to source code and frequently referenced it.

The linking of screencasts to external resources demonstrates that knowledge is held in many different resources. Distributed Cognition is the theory that cognitive processes are spread across a number of artifacts [43]. For example, a user many rely on to-do lists or written information, to support understanding a program [85]. In a

screencast, these sources of information are not only contained in the program's code, but also within Websites, APIs and other documentation. The developer is also a source of information for the audience.

The theory of distributed cognition is referenced because screencasts act as an artifact for developers to store knowledge. By creating a screencast, developers are also storing their cognitive process within a video. They make references to source code, diagrams and other documentation, which supports the development of a mental model.

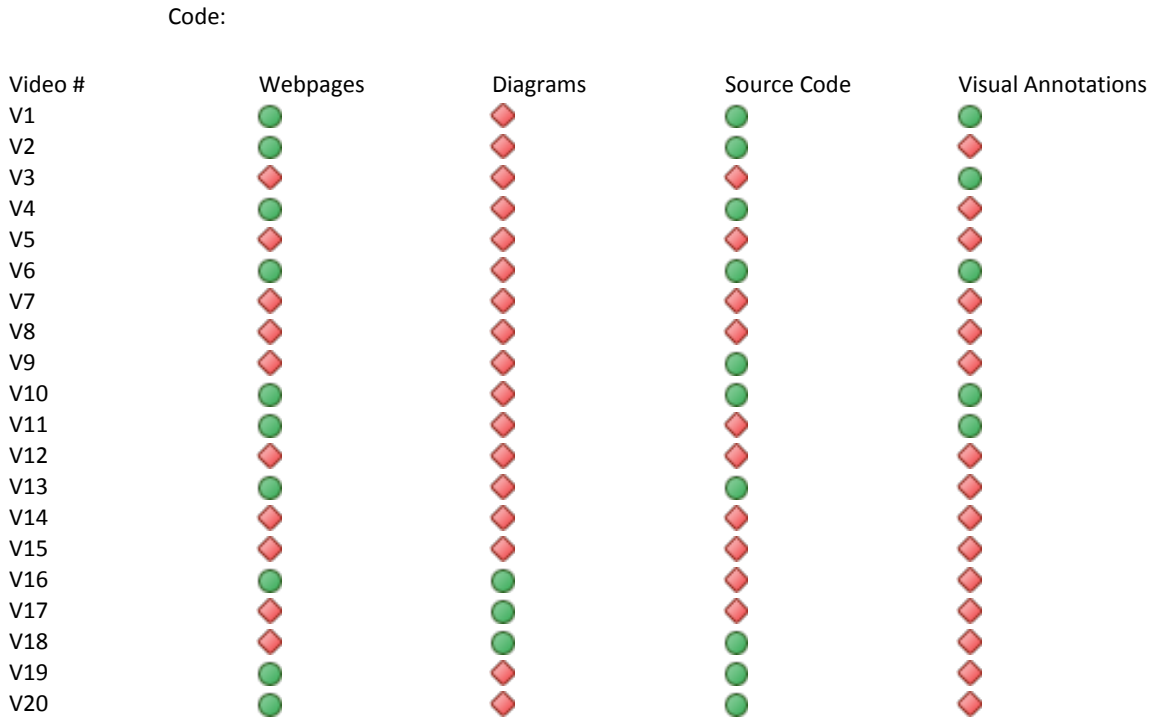


Figure 5.6: The results from coding the set of videos for the additional resources theme.

5.3.7 Mapping Execution to Code and Code to Code

Mapping plays an integral role in the screencasting process. It is a way of logically moving through complexity, in order for the audience to make sense of the code. Linking together segments allows developers to connect concepts together 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 dif-

ferent abstraction levels...by mapping program parts to functional descriptions" [84]. By linking beacons, narrators navigate the program's code.

Narrators were observed creating mappings between features and segments of software. Narrators were observed **making connections between the execution and the code**, to demonstrate where a feature resided$_{V2,V5,V6,V9,V13,V15,V16,V17,V18,V19,V20}$. They also moved between **segments of code** to explain relationships $_{V1,V2,V3,V5,V7,V9,V10,V11,V12,V13,V15,V16,V17,V18,V19,V20}$. In the screencasts, narrators used these mapping techniques to connect concepts within the program.

Building on the previously mentioned beacons, Brooks hypothesizes that text and documentation serve as a "collection of indicators to be assembled into beacons" which developers assemble to form their understanding of a program [12]. This mapping builds not just between parts of the text, but also on the conditional implementation of the text. As described previously, screencasts let the narrator reference the control flow and structure of a program. By looping these beacons together, a narrator shares their comprehension of the code.

Figure 5.7: The results from coding the set of videos for the mapping execution to code and code to code theme.

Developers use a variety of techniques to communicate knowledge through screencasts. 20 YouTube screencasts were analyzed to understand the techniques developers' use and from this analysis developed a set of themes. It was found that developers define the purpose of their videos, talk about code in various levels of detail and explore the project's structure. They also share information about how to adapt their programs through demonstrations, live editing, and by providing resources.

Chapter 6

Discussion

This chapter discusses the screencasts studied for this thesis in relation to the literature and the observations. Based on the literature and findings, the chapter presents a number of guiding best practices for creating screencasts aimed at developers, the place of YouTube in the social developer ecosystem and the challenges of screencasts as a medium.

6.1 Screencast Best Practices

Based on the observations, a list of best practices for developers wishing to create screencasts was developed. Interviewees were asked what they thought were good and bad characteristics of screencasting. This line of questioning heavily influenced the list below.

Screencasting Best Practices	
1.	Plan Ahead
2.	Short and Focused
3.	Provide the Source Code
4.	Speak Clearly
5.	Execute the Code

Table 6.1: A list of best practices developed from the findings of this study and the literature.

Plan Ahead: As shown in Chapter 4, 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 [57]. Planning helps tailor content for the audience's, "level of knowledge" [61]. Previous work suggests that noise, or confusing information, be filtered out at this stage [61].

Short and Focused: The interviewees spoke of the need for short and clear screencasts. They discussed the importance of not rambling, and that screencasts need to be on topic$_{P02,P05,P06,P08,P10}$.

The educational literature suggests "minimizing the memory load" when creating screencasts [61]. This means not overwhelming the audience with too much information. Therefore, screencasts should be to the point and on topic.

As previously mentioned, the interviewees felt that it's easier to edit or re-record short video segments. Interestingly, the videos in my sample ranged from 3 minutes to over an hour. Just because interviewees stressed the importance of brevity does not mean that they practiced it:

> "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 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." $_{P02}$

Provide the Source Code: Developers stressed the importance of providing source code to the audience$_{P03,P04,P08,P09}$. This reflected what was observed in the screencast analysis. These screencasts provided links to source code and frequently referenced it. From the literature, it is known that "learning enough to apply new knowledge usually requires active engagement or practice in a realistic context" [61]. The source code provides a real example which the audience can use to experiment and build upon.

Speak Clearly: All of the videos in the screencast sample contained narration. Interviewees stressed the importance of being able to understand the person speaking in the screencast. Narration allows for both explicit and implicit description of the actions on the screen [79]. These audio clues, coupled with visual representations, help audience members learn and internalize information from the screencast [61].

Execute the Code: As shown in Chapter 4, nearly all of the videos in the screencast sample showed the program under discussion being executed. Audience members need to make connections between the program and the source code. According to Brooks, executing the code allows the audience to create mappings between the code and the problem it solves [12]. Previous work also suggests that it provides context, and sets audience expectations as to what the program does [61].

These best practices address common concerns that were brought up in the interviews with screencast creators. Unlike the previous cited educational literature, there are a number of challenges unique to the developer screencasting experience that these practices address. Unique to software development, screencast creators must be aware of the possibility of programming errors and formulate strategies to deal with them. This study shows that developers have found inventive ways to take these errors and turn them into teachable moments. Through planning ahead and executing the code, developers can mitigate or incorporate these bugs into their screencasts.

Additionally, creating a screencast requires domain specific programming knowledge. This extends not only to programming languages, but also to programming tools. Developers can use screencasts to share their insights into multiple aspects of programming. They can do this in screencasts by capturing footage of tool use, providing source code for the audience to follow along with, and by executing the code for demonstrations. By providing the audience with a clear, well-planned video, it is hoped that these guidelines provide support to future screencast creators.

6.2 YouTube in the Social Developer Ecosystem

YouTube was used by interviewees to share insights, and to build an online reputation. In alignment with the literature, the interviewees reinforced that developers shape their online identity by choosing to share content [70]. This was done for a number of reasons, including altruistic and economic motivations.

From this study, it was found that developers use YouTube to supplement their activities on other platforms. For example, interviewees spoke of how social media impacted their screencast creation process. Many described how being active on other social media channels provided them with insights for their content$_{P01,P02,P03,P04,P06,P07,P09,P10}$. In this way, interviewees remained aware of programming-related news and developments.

From these insights, we know that the interviewees used YouTube in conjunction with other social media platforms. Six of the interviewees referenced other screencast channels, which they used as inspiration for their own work$_{P02,P04,P05,P06,P07,P08}$. Interviewees indicated that they draw ideas from different sources to create social media content. In addition, interviewees described how their previous work and audience interactions gave them inspiration for future content$_{P02,P04,P06}$. For example, as P02 described,

> "A lot of it comes out of someone requesting for help, where I'll go on later on down the road to make those videos, I have planned to do a couple others that I've gotten requests for."

The degree to which interviewees received, and interacted with, this feedback varied. One developer explained that he stopped monitoring comments on YouTube after they became overwhelming$_{P08}$. Others expressed disappointment at the lack of feedback they received$_{P04,P05}$. Those who I interviewed were surprised by the positive feedback they received, which congratulated them on their videos, or expressed how helpful they had been$_{P04,P06,P07,P08,P10}$. Interviewees described that this provided them with validation for their work$_{P06,P07}$. Interviewees also received feedback from people reaching out for programming help$_{P01,P02,P05,P08}$, but reported minimal negative feedback or trolling$_{P01,P04,P07,P08}$.

YouTube allows the audience a way of interacting with the content creator. As the interviews showed, screencast creators become a contact point for struggling learners. One can see that YouTube is used in conjunction with other media platforms to connect developers, and share content.

6.3 Limitations of Screencasts

As a medium, there are a number of limitations surrounding the use and creation of screencasts. Screencasts are limited by the effort required to update or alter them, the platforms on which they are hosted and the degree to which they help users locate relevant information.

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 reshoot and add in any missing information. Reshooting requires recreating the technical environment, project and code structure shown in the screencast. There

could be many scenarios where this is not feasible or possible. Therefore without action on the creator's part, screencasts may be incomplete or out of date.

In order to reach a large audience, screencast creators are reliant on the use of online video sharing platforms. While YouTube is arguably the most popular of these, there are many video sharing platforms on the Internet. These platforms operate with their own rules and requirements for posting and accessing video content. This leads to variations in the accessibility and permanence of screencasts, which affects who and how audience members access online video. Screencasts are therefore limited to where the creator posts them online, and the terms of use for that specific platform. For example, currently YouTube requires users to have an account if they want to post comments in response to a video. This limits and shapes the interactions screencast creators have with their audience.

Additional, information openly available on the Internet is no help to anyone if it cannot be found. Unfortunately video itself is not easy to search. This means audience members must watch them to determine if the video suits their needs. For long videos, this is not efficient. Screencasts and videos as knowledge resources are limited in that they require this additional data in order to communicate their contents to the audience.

Video sharing platforms help audience members find the correct content through the use of a number of features like search, tags and creator supplied meta data. Meta data can outline the purpose and contents of a screencast to the audience. I observed creators using meta data to point audience members to additional resources, like source code and written tutorials. The interviewees stressed that in addition to meta data, creating focused videos is very important in helping audience members find relevant information.

Finally, the participants frequently expressed frustration with the tools they used to create screencasts$_{P01,P04,P06,P07,P08,P09}$. Sources of frustration included editing and recording software, upload times, video quality, and adjusting to screen resolutions. Interviewees reported that they spent the most amount of time in post-production improving the quality of their work$_{P01,P04,P07}$.

From the interviewees, it was identified that a large amount of the screencast creator's time is spent in post-production. Interviewees expressed frustration with the tool support for creating screencasts. From this work, it has been identified that there are opportunities for improvements to better support this type of work.

This thesis has looked at YouTube videos "in the wild", created by developers with informal learning objectives. From this work, it has been shown that YouTube is used by developers to share information through video. Not only does video allow developers to explain the functionality of their program, but it also allows them to share their programming environment and style. This can be done through techniques such as debugging, and showing the tools they use for software development.

Screencasts can be used to share information, but this work has also highlighted the contemporary limitations of screencasting. For example, the interviewees demonstrated how updating screencasts is a cumbersome task. Based on the findings of this work, it has been found that there are also technological limitations to screencasts that make it difficult to locate information.

Chapter 7

Threats to Validity and Limitations

In this section, the threats to the validity of this study are outlined. This chapter also discuss the challenges of applying a grounded theory methodology by showing how best practices from the literature were incorporated into the approach of this study.

7.1 Internal Threats to Validity

Internal validity is concerned with the extent of bias within a study [26]. The researcher's actions and biases play a large role in the internal validity of a grounded theory project. Therefore, there are threats to the quality of the work that the researcher must be watchful for throughout the process.

In this study, my biases may have impacted my analysis of the data. The background of those involved in analyzing the study's data was outlined in Chapter 3. I reiterate though, that, as researchers, my collaborator and I have an English speaking background with educational experience in computer science. This background might have biased who, and what, was chosen for this study.

The data selection methods (as explained in Chapter 3) could have introduced selection bias. A systematic YouTube search was conducted for this study, paired with a set of selection criteria. The search terms used impacted the videos represented in the set. For example, the search terms "Walkthrough" and "Tour" were chosen to reflect the line of questioning at the time, which focused on understanding walkthrough methods. This influenced the videos that appeared in the search results and therefore the data set. The search results were ranked by popularity. Choosing screencasts based on date, or including different search terms, would have affected

the outcome of this work. The data set is therefore biased towards popular videos, as ranked by a standard YouTube search.

This work relied on YouTube for the data set, which contains a bias for videos showing programs with a graphical element. For example, a number of screencasts in the data set focused on programs with user interfaces. Other types of programs might be difficult to present through video, and are therefore not represented in the sample. Given the number of videos on YouTube, there is no way to know how representative the sample is. Regardless, the codes and themes were well saturated after 20 videos, which provides confidence in the sample.

In analyzing the screencasts, a percentage agreement calculation was used to determine inter-rater reliability. As previously noted, using inter-rater reliability is a quantitative method, and not traditionally used in grounded theory. Using a percentage agreement calculation has its own limitations, which were discussed in Chapter 3. The calculation was a methodological guide for the study, not a statical finding, and therefore, deemed appropriate for the study's goals. A similar use of this approach can be found in previous work by Creswell [25].

The methods used to contact developers for interviews, could also have been influenced by biases. Those contacted had accounts in the screencast sample which had been recently active, or did not belong to a large, multi-national corporation. While initially accounts with these characteristics were contacted, no responses were received from accounts with these characteristics and the approach was refined to exclude them. It was assumed that these accounts would not be available for the study. Individuals were contacted through their YouTube accounts and publicly listed contact information.

Those who create screencasts are assumed to have a more positive attitude towards screencasts. Previous work shows that content creators typically make up the minority of a social media platform [54]. Therefore, one cannot assume that the views of the interviewees in this study represent the majority of YouTube users. The interviewees might hold a unique view of screencasting and of YouTube. This might have biased their responses, ultimately affecting the findings on screencasts for developers.

7.2 External Threats to Validity

External validity is the extent to which the findings of a study can be generalized to other populations or settings [14]. It is a known limitation of grounded theory that

the theory cannot be considered generalizable, until further evaluations have been conducted [44].

Grounded theory is concerned with exploring the experiences of those being studied [17]. For this reason, the findings are only applicable to the set of videos and interviews analyzed in this thesis. The findings of this study are not generalizable to the population of all developers who create screencasts. This does not mean that work cannot be done to determine to what extent the findings apply to a larger population.

7.3 Common Pitfalls of Applying Grounded Theory Work

As Charmaz describes, sometimes the, "weaknesses in using the method may become equated with weaknesses inherent in the method..." [16]. While grounded theory has been described as a deceptively simple methodology [3], to do it well requires constant vigilance and awareness of the data. There is a large body of work addressing common mistakes from researchers applying the method, including work by Suddaby [78], Adolph et al. [3], and Jones and Alony [44]. This section will highlight some common issues, and address how we overcame them.

7.3.1 The Role of the Researcher

Because the researcher's role is to interpret the data, Creswell writes that they must put aside preconceived ideas and come to the data with an unbiased mind [26]. Charmaz's opinion is that theory is "constructed" by the researcher through their, "past and present involvements and interactions with people, perspectives and researcher practices" [17]. The researcher is never a blank slate, but by acknowledging their biases the literature suggests that they can disclose ideas that might influence their work [78].

The literature suggests that to combat this problem one must be cognizant of their preexisting knowledge of the subject area, which I disclosed in Chapter 3 [78].

Researcher bias is another reason why some suggest doing the literature review after data analysis [17]. While I was familiar with the program comprehension literature before the start of this study, the literature review woven into the findings of this paper occurred after the data collection and analysis phases. Furthermore, literature on education, knowledge sharing, and screencasts was woven into the thesis after the

findings of the paper had been established. In this way, attempts were made to limit the influence of the literature on interpreting the data.

7.3.2 Coding and Collecting Data

A common mistake in grounded theory is committing to a set of codes or categories "prematurely" [16]. By doing this, the researcher jumps to conclusions, and stops being open to the data [78]. This is a danger of letting preexisting biases influence the data analysis.

Through the use of team-based coding, an effort was made to reduce the degree of bias in the data analysis. Both my colleague and I held our own set of experiences and biases coming to the study. When we began coding the screencasts, I told the secondary researcher to code, "what they saw" in the screencasts. This description was intentionally vague and left open to interpretation, and the phrase shaped our coding process. With both the screencast and interview data, we began by exploring the data alone, before coming together to discuss our ideas. We did not always agree, and spoke through our different interpretations of the screencasts.

7.3.3 Presenting Grounded Theory Work

Grounded theory work is presented to the reader as a linear process. This is a known issue in the literature [41]. For the writer, there is a tension between presenting the methodology in a way that is understandable, but still truthful to the disjointed process.

This thesis aims to present the process as clearly as possible, and avoid confusing the reader. The descriptions and diagrams in Chapter 3 give an idea of how we analyzed the data multiple times to redefine codes and themes. Discussions and thoughts about the codes were recorded, which allowed us to reflect upon our ideas over time. As researchers, we repeatedly pushed ourselves to explore the data in new ways. Multiple methods were used over time to conduct this grounded theory study. It was not a linear process, but an emerging exploratory study.

This chapter has outlined threats to the validity of this work. The main threat to this work is the biases introduced by the researchers. The limitations of this work stem from the participants interviewed and the generalizability of the findings. To

address these issues, future work must be done to explore the experiences of other screencast creators.

Chapter 8

Future Work

In studying how software developers use YouTube, this work raises a number of questions for future work. It has been shown how developers create screencasts and share content via social media. Based on the findings, this work identified the practices and tools used by a subset of screencast creators. The following subsections outline areas for future work, based on the findings.

8.1 To what extent are screencasts used by developers?

The participants reported that they received feedback from their audience, through a number of channels. This feedback not only validates the screencast creator's work, but also helped them gauge the degree to which people found their content useful. By looking at the view counts of YouTube videos, one can infer that people watch these screencasts. What is not known is how many people watch these types screencasts, and how this type of information is integrated into the development process. To address this question, survey techniques can be used to understand how, on a large-scale, developers use screencasts. In order to understand how developers integrate this information, one could us observational techniques and interview sessions.

8.2 Are screencasts effective?

Building on the previous question, another line of inquiry that emerged from the study was to what extent developers find screencasts effective. Specifically, for what goals

and tasks do they find them effective? By exploring the effectiveness and usefulness of screencasts for developers, one can build on the screencast techniques identified in this thesis. This would allow us to evaluate their appropriateness for screencast audiences. This line of questioning could also be investigated via survey, observational and interview techniques. Developers could be asked for examples of effective screencasts and what situations they are best suited for.

8.3 What type of comments do screencasts receive on YouTube?

While collecting screencasts, it was observed that viewers used YouTube's commenting feature to interact with the screencast creator. This was also noted in the interviews: interviewees reported receiving comments and questions from audience members through YouTube.

Audience members write a comment to ask questions related to the screencast. This suggests that comments play a role in helping the audience utilize screencasts. Future work should explore how the commenting feature is used by audience members and content creators, to understand how both types of users interact. This research would help us understand the type of support viewers receive through this commenting structure, and what questions users ask. This could be studied through an analysis of YouTube video comments and follow up interviews with commenters and screencast creators. Comments on other video platforms could also be analyzed to explore any differences between the cultures of users on these hosting sites.

8.4 What communities are formed around screencasts?

Building off the previous question, it was also observed that screencasters interacted with their audience through YouTube's commenting features. From the interviews, we know that that these interactions create relationships. 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[1]. A number of the interviewees also spoke of popular developer-focused screencasting sites that influenced their work.

From this, it is known that there are communities built around producing and sharing developer knowledge through screencasts. In the future, work may wish to explore these emerging communities and the relationships between content producers and audience members. Ethnographic and other qualitative methods can be used to better understand these communities.

8.5 How do developers create screencasts?

The interview findings are based on self-reported data. In future studies, work should be done using real world observation techniques to understand the process behind creating screencasts. This would further establish the techniques used by developers and the friction points they encounter in the process. A study of this would allow us to understand the developer's thought process when creating a screencast.

8.6 What is the current tool support for creating screencasts?

As reported in the findings, interviewees expressed frustration with their current tool support. For this study 10 screencast creators were interviewed, which is a small sample size. So, this findings may not reflect the actual practices of screencast creators on a larger scale.

In order to help screencast creators further though, one must understand their current state of tool support. Identifying the current tools being used and the tool needs of developers, would be one approach for better understanding the current state of practice. This can be done through collecting data from screencast creators about what tools they use, and analyzing the features of these tools.

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

8.7 What are the tool requirements for developer-focused screencasting tools?

After one understands the current tool support, future work could contact screencast creators to understand what requirements a screencast creation tool needs. One could also dig deeper into the friction points they experience.

By understanding these tool friction points, future work can create a list of tool requirements for screencast creation software. An area for future work should look at applying the lessons learned in this study to create tool support requirements. The hope of these tool requirements would be to inform future screencast creation software.

8.8 Understanding program comprehension models and screencasts

In order to create a screencast, the creator must understand the program. To do this, they need to create their own mental model. By explaining the software, they explain their mental model of the program. Therefore, the screencasting literature shares commonalities with the study of program comprehension.

In the program comprehension literature, there are a number of models, which explain how developers understand software [84]. In future work, it is hoped that a set of screencasts can be analyzed using these model to evaluate the role of mental models in creating screencasts.

As an outcome of this study, there is a better understanding of how and why developers create screencasts for YouTube. The exploratory approach raised multiple questions for future work. There are interesting questions about how these screencasts are used by audience members, as well as how the creators can be better supported through software and tools.

Chapter 9

Conclusions

This thesis began by exploring screencasts for developers hosted on YouTube. Software development is an intellectually challenging task, and screencasts serve as a way to document knowledge. Coupled with audio, screencasts allow people to provide a rich description of the content on their screens. The YouTube platform situates these videos in a social ecosystem, and provides content creators with mechanisms for community engagement.

Developers use these videos to share their software development experiences. From the analysis in this thesis it was shown that developers create videos on a range of topics. These videos demonstrate how to solve novel problems. By hosting these videos on YouTube, developers disseminate information and contribute to a larger selection of content. The videos encapsulate knowledge, and serve as a persistent record of the developer's insights. Because they are tied to the creator's online identity, the videos become a part of the creator's presence on social media platforms.

This work adds to the literature on software development and screencasting practices. Through analysis, I explored how developers use video to share knowledge. This work looked at the specific techniques they used in the YouTube videos, and categorized those techniques.

The videos studied represent artifacts for developers to engage in spontaneous learning. This refers to learning that takes place outside of a formal classroom, in a real world setting. This work focused on the unique developer experience for sharing domain specific information through screencasts, and how these experiences differ from the existing screencast literature.

The findings showed many similarities to the screencast literature, but also observed developer specific challenges. For example, bugs in computer programs created

complications for developers while filming. These bugs can be tailored to be teachable moments for the audience, or seen as errors in the video production process that need to be addressed. However, in order to create screencasts, developers need to have domain specific knowledge of tools and languages, which they share during the video creation process.

This research also addressed why developers created these screencasts, and therefore interviewed 10 screencast creators. From the interviews, they suggest that developers create screencasts because of a number of motivating factors. One of these was for personal benefit or to give back to their community. Emergent details suggest that screencasters tailor their content for their audience, and that some prefer screencasts over blogs. For the screencast creation process, this work suggests that the interviewees experienced a level of frustration with their current tool support.

From the interviews and observations, a list of best practices for screencast creation was put forth. The findings stressed the importance of short, focused videos, as this helps overcome some of the limitations of video.

The main contribution of this work is a first look at how YouTube is used by developers to share knowledge. Current screencasts aimed at developers use a number of different approaches to communicate information, and exist in a complex, social setting.

Appendices

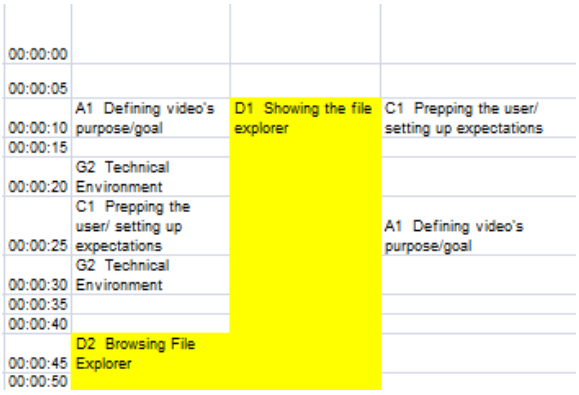
Appendix A

Codebook

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

Explanation

Videos will be coded by the researchers in 5 second increments using a spreadsheet. Multiple codes can occur at once. If an instance of a code continues for a length of time, the coder will note this in the table. Below is an example of our earlier coding using older codes.



00:00:00			
00:00:05			
00:00:10	A1 Defining video's purpose/goal	D1 Showing the file explorer	C1 Prepping the user/ setting up expectations
00:00:15			
00:00:20	G2 Technical Environment		
00:00:25	C1 Prepping the user/ setting up expectations		A1 Defining video's purpose/goal
00:00:30	G2 Technical Environment		
00:00:35			
00:00:40			
00:00:45	D2 Browsing File Explorer		
00:00:50			

Codes

Goal setting

Defining the video's purpose

- Definition: The presenter explains to the audience what the goal of the video is. What task or program is the presenter trying to show to the viewer? The goal should be explicitly stated. Typically the goal will be stated in the beginning of the video. Think of this as the definition or topic of the video, the problem the speaker is going to address.
- Example:

> *In this video I'm going to show you the simple chat example that comes with the Electro server...*
>
> *-Simple Chat Walk-through with Unity*

Explaining the video's limits and audience

- Definition: The presenter is telling the user what they are going to see in an upcoming segment of the video. Could be something they will see in the next two seconds or minutes from now. Importantly, the presenter is setting limits for the viewer, indicating what they may be leaving out of the film or skipping over.
- Example:

> *So, I'm just going to show you the Electro Server API pieces and, um, pretty much just skip over any of the unity stuff that you probably already know*

-Simple Chat Walk-through with Unity

Live editing

Making live code changes

- Definition: When the presenter makes changes to the body of code during the video.

Changing control flow variables

- Definition: When the presenter alters the logic or flow of the code during the video.

Introducing bugs

- Definition: When the presenter makes a code change in order to introduce a bug. Can be explicitly or accidentally done.
- Example:

Demonstrations to showcase the execution of the program

Executing the program to demonstrate features to the audience

- Definition: Occurs when the presenter runs the program to communicate to the viewer what the expected output or resulting functionality is. Shows the user the end product.

Referencing different levels of detail

High level code description

- Definition: Talking about the purpose of code, without discussing the technical details. Abstracting away from the low level code.
- Example:

> *"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." from Code Walkthrough - Cafe Townsend Robotlegs for Corona SDK*

Medium level code description

- Definition: Functional discussion of more than one line of code. Can be a technical discussion of a sub block of code.
- Example: "First I want to make sure that the directory exists for where we are going to install [the launcher]." from Minecraft Launcher - Code Walkthrough

Low level code description

- Definition: When the presenter describes the functionality of a specific line. Think of it as the opposite of a high level code purpose.
- Example: The presenter said, "There is also a current room property with the data type of room and it's set to null" when reading the following code out loud:

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

Pointing out element identifiers

- Definition: Depending on the language an identifier may be a class, function, or variable name. I1 is used whenever the presenter references a specif <variable, method, type> in the code base. They highlight this identifyer for a specific reason. _Note: _
- Example: `a = foo.bar('43')` If the presenter makes a point of describing a, foo, bar and "43", then it is an instance of this code.

Referencing line number

- Definition: The presenter includes the line number in their specific verbage. Not to be confused with explaining the functionality of a line.

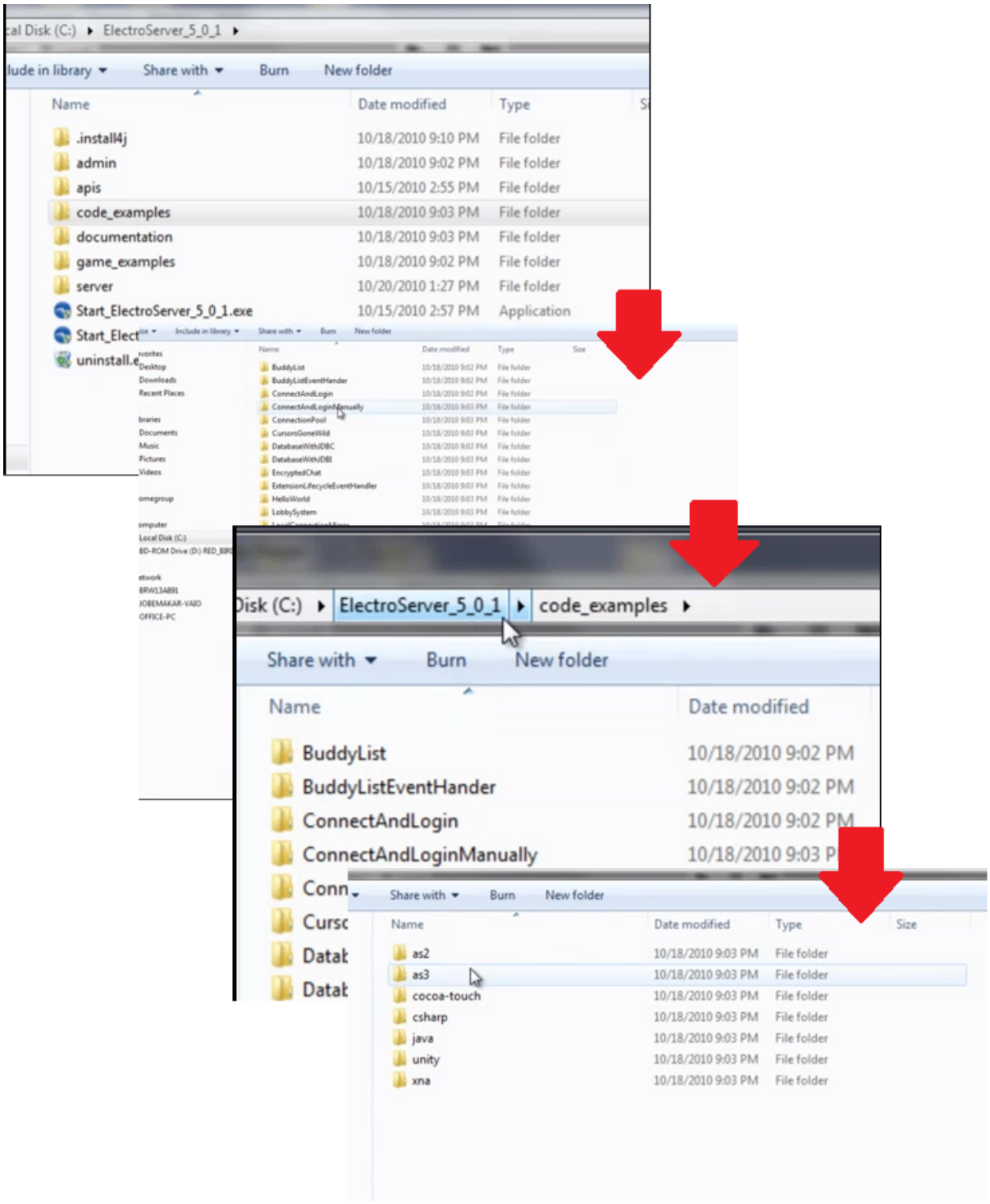
Referencing return types and parameters

- Definition: When the presenter explains or mentions the return value of a function or the needed parameter of a function.

Browsing the technical environment

Making use of the file explorers

- Definition: The presenter navigates around the file explorer showing folders or files relevant to the project being presented. Imparts a sense of the project structure onto the viewer. Also provides them with knowledge of where to find resources and important files. When using this tag, indicate the length of the task.
- Example:



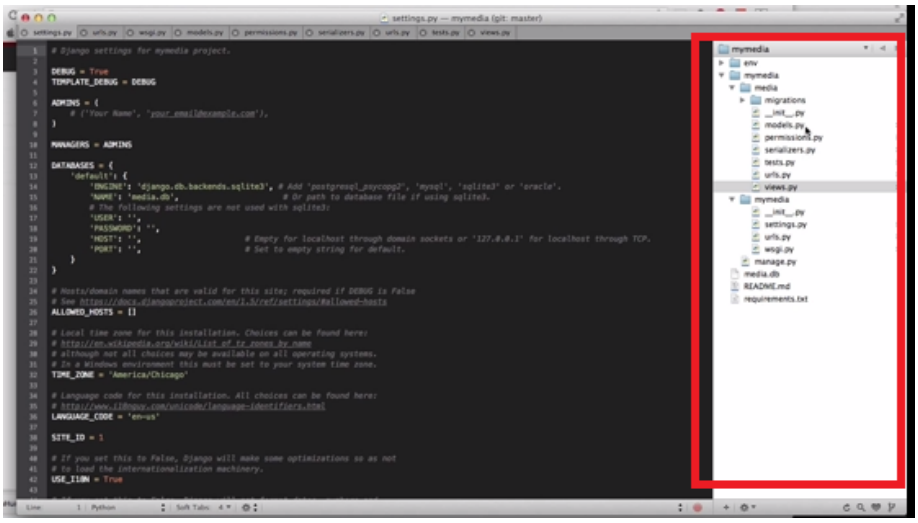
Explaining the program structure

- Definition: Used when the presenter explains or mentions the project setup to the viewer. This could include showing the folder layout in an IDE.

- Example:

> *When you first go through the tutorial, there is a couple of commands in Django that will actually create a skeleton for an application, and basically the skeleton is what you see here. I don't think I added any files...*

-Django - high level walkthrough



Explaining the technical environment

- Definition: Many times, running a program or programming in a particular language requires some technical setup of the environment. This code refers to those instances. Used when the presenter shows or references a prerequisite for the program to run or to be developed.
- Example: Software Libraries, IDE, Server installed and running or explaining what dependencies are needed for the project.

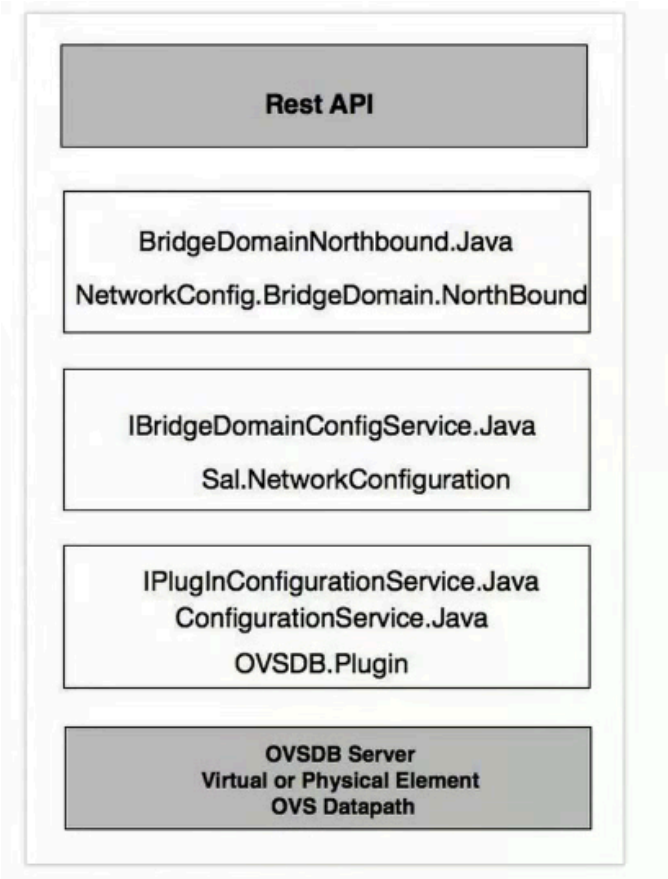
Provisioning of additional resources

Webpages

- Definition: Narrators may mention or explicitly link to other webpages where the audience can find relevant information. This includes supplemental materials and written versions of the screencast.

Diagrams

- Definition: Visual images used to explain the program to the audience
- Example:

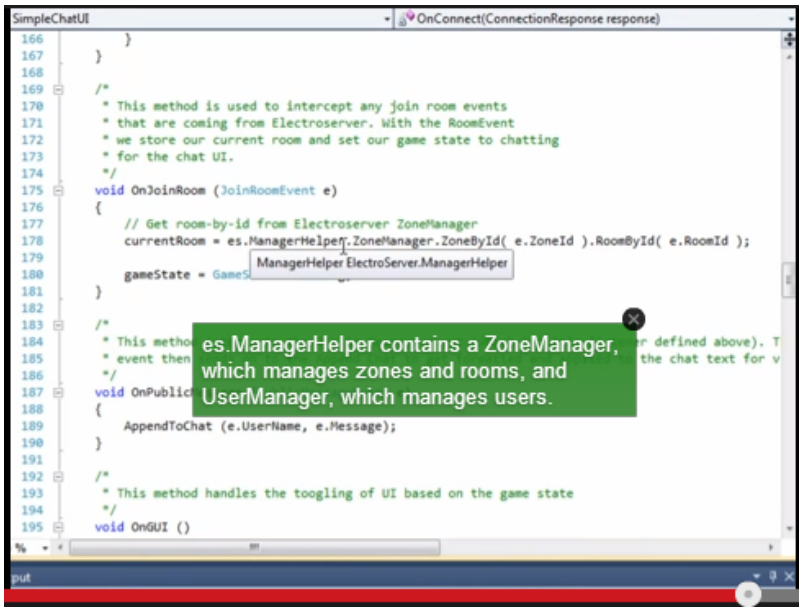


Source code

- Definition: Sometimes the creator of the video will include a reference to the source code being shown in the video. This code describes when the source code is made available to the audience. This could be through GitHub, Dropbox or some other method.

Visual annotations

- Definition: Typically textual boxes added to the video in postproduction to supply the user with additional information about either to project or the developer. Does not include advertisements of YouTube.
- Example:



Mapping execution and code to code

Connection between the demonstration and code

- Definition: The presenter references a demoed or not demoed feature of the final running application that is directly related or effect by the code segment.

Linking code segments together

- Definition: When the presenter is discussing identifiable segments or lines of code that are separated, and one of which is in the view of the presenter. The separation does not have to be above or below the code since the linked segment could be located in another file.
- Example: "This code is connected to this code"

Appendix B

Semi-Structured Interview Questions

1. Tell me a bit about yourself.
2. What do you do for a living?
3. (If not obvious from the above:) Would you consider yourself a developer?
4. Where are you located?

5. As a contributor how do you use YouTube?
6. How many videos do you have on YouTube?
7. What prompted you to start posting on YouTube?

8. How do you decide to make a video?
9. Think about the last video you made, can you describe the process?
10. Were these any different than other videos you have made?
11. If you have many videos, has your process changed over time?

12. Who do you think your audience is?

13. How do you picture them?
14. What do you think your audience is trying to get out of your videos?

15. What tools/ software do you use to make your videos?
16. Do you edit your videos?
17. If so, how long does it take?

18. Have you received any feedback from your videos?
19. If so what has it been like? What have people said?
20. What do you think makes a good code walkthrough video?

Appendix C

YouTube Video URLs

1. https://www.youtube.com/watch?v=0d-wY65uVGw
2. http://www.youtube.com/watch?v=6MpuB654_hM
3. http://www.youtube.com/watch?v=UiST0tcOWvU
4. http://www.youtube.com/watch?v=qUpiWWjOfRw
5. https://www.youtube.com/watch?v=nwn3YY6cyEQ
6. https://www.youtube.com/watch?v=djApduemlf4
7. https://www.youtube.com/watch?v=jGR0EVYc_Bo
8. https://www.youtube.com/watch?v=mn_wW8uZ6eQ
9. https://www.youtube.com/watch?v=G1DbLOVs7UM
10. https://www.youtube.com/watch?v=0npv906IQag
11. https://www.youtube.com/watch?v=wg_gNs3Xxq4
12. https://www.youtube.com/watch?v=UdDr9QquiLc
13. https://www.youtube.com/watch?v=T1-yARGKhXU
14. https://www.youtube.com/watch?v=kwY-8mAyixU
15. https://www.youtube.com/watch?v=WCuQOjD8w_E
16. https://www.youtube.com/watch?v=T1f4xXoRDG

17. https://www.youtube.com/watch?v=3-jCTvNRJS0
18. https://www.youtube.com/watch?v=OPuYYLEyz-A
19. https://www.youtube.com/watch?v=iep-z1KXRN8
20. https://www.youtube.com/watch?v=O130d8ioFS4
21. https://www.youtube.com/watch?v=wIodpWdvqaU

Bibliography

[1] Grounded Theory: An Exploration of Process and Procedure, author=Walker, Diane and Myrick, Florence. *Qualitative Health Research 16*, 4 (2006), 547–559.

[2] Adamic, L. A., Zhang, J., Bakshy, E., and Ackerman, M. S. Knowledge Sharing and Yahoo Answers: Everyone Knows Something. *Proc. of the 17th Intl Conference on World Wide Web* (2008), 665–674.

[3] Adolph, S., Hall, W., and Kruchten, P. A Methodological Leg to Stand on: Lessons Learned Using Grounded Theory to Study Software Development. *Proc. of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds* (2008), 13.

[4] Adolph, S., Hall, W., and Kruchten, P. Using Grounded Theory to Study the Experience of Software Development. *Empirical Software Engineering 16*, 4 (2011), 487–513.

[5] Adolph, S., Kruchten, P., and Hall, W. Reconciling Perspectives: A Grounded Theory of How People Manage the Process of Software Development. *Journal of Systems and Software 85*, 6 (2012), 1269–1286.

[6] Agazio, J., and Buckley, K. M. An Untapped Resource: Using YouTube in Nursing Education. *Nurse Educator 34*, 1 (2009), 23–28.

[7] Allan, G. A Critique of Using Grounded Theory as a Research Method. *Electronic Journal of Business Research Methods 2*, 1 (2003), 1–10.

[8] Begel, A., DeLine, R., and Zimmermann, T. Social Media for Software Engineering. *Proc. of the FSE/SDP Workshop on Future of Software Engineering Research* (2010), 33–38.

[9] Biggerstaff, T. J., Mitbander, B. G., and Webster, D. The Concept Assignment Problem in Program Understanding. *Proc. of the 15th Intl Conference on Software Engineering* (1993), 482–498.

[10] Brandt, J., Dontcheva, M., Weskamp, M., and Klemmer, S. R. Example-Centric Programming: Integrating Web Search into the Development Environment. *Proc. of the SIGCHI Conference on Human Factors in Computing Systems* (2010), 513–522.

[11] Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., and Klemmer, S. R. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. *Proc. of the SIGCHI Conference on Human Factors in Computing Systems* (2009), 1589–1598.

[12] Brooks, R. Towards a Theory of the Comprehension of Computer Programs. *Intl Journal of Man-Machine Studies 18*, 6 (1983), 543–554.

[13] Burgess, J., and Green, J. *YouTube: Digital Media and Society Series*. Polity, Cambridge, 2009.

[14] Calder, B. J., Phillips, L. W., and Tybout, A. M. The Concept of External Validity. *Journal of Consumer Research* (1982), 240–244.

[15] Capiluppi, A., Serebrenik, A., and Singer, L. Assessing Technical Candidates on the Social Web. *IEEE Software 30*, 1 (2013), 45–51.

[16] Charmaz, K. Discovering Chronic Illness: Using Grounded Theory. *Social science & Medicine 30*, 11 (1990), 1161–1172.

[17] Charmaz, K. *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. Sage, London, 2006.

[18] Charmaz, K., and Belgrave, L. Qualitative Interviewing and Grounded Theory Analysis. In *The Sage Handbook of Interview Research: The Complexity of the Craft*, J. F. Gubrium, Ed., vol. 2. Sage Publications, London, 2002.

[19] Chenail, R. YouTube as a Qualitative Research Asset : Reviewing User Generated Videos as Learning Resources. *The Qualitative Report 1*, 4 (2011), 18–24.

[20] Chtouki, Y., Harroud, H., Khalidi, M., and Bennani, S. The Impact of YouTube Videos on the Student's Learning. *2012 Intl Conference on Information Technology Based Higher Education and Training* (2012), 1–4.

[21] Coleman, G., and O'Connor, R. Using Grounded Theory to Understand Software Process Improvement: A Study of Irish Software Product Companies. *Information and Software Technology 49*, 6 (2007), 654–667.

[22] Coleman, G., and OConnor, R. Investigating Software Process in Practice: A Grounded Theory Perspective. *Journal of Systems and Software 81*, 5 (2008), 772–784.

[23] Cornelissen, B., Zaidman, A., Van Deursen, A., Moonen, L., and Koschke, R. A Systematic Survey of Program Comprehension Through Dynamic Analysis. *IEEE Transactions on Software Engineering 35*, 5 (2009), 684–702.

[24] Creswell, J. W. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Sage Publications, London, 2009.

[25] Creswell, J. W. *Qualitative Inquiry and Research Design: Choosing Among Five Approaches*. Sage Publications, 2012.

[26] Creswell, J. W. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Sage Publications, 2013.

[27] Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. *Proc. of the ACM 2012 Conference on CSCW* (2012), 1277–1286.

[28] Détienne, F. *Software Design–Cognitive Aspect*. Springer, London, 2002.

[29] DiMicco, J., Millen, D. R., Geyer, W., Dugan, C., Brownholtz, B., and Muller, M. Motivations for Social Networking at Work. *Proc. of the 2008 ACM Conference on CSCW* (2008), 711–720.

[30] Doering, E., and Mu, X. Circuits Learned by Example Online (CLEO): A Video-Based Resource to Support Engineering Circuit Analysis Courses. *Frontiers in Education Conference* (2009), 1–4.

[31] Duffy, P. Engaging the YouTube Google-Eyed Generation: Strategies for Using Web 2.0 in Teaching and Learning. *European Conference on ELearning* (2007), 173–182.

[32] Duncan, I., Yarwood-Ross, L., and Haigh, C. YouTube as a Source of Clinical Skills Education. *Nurse Education Today 33*, 12 (2013), 1576–1580.

[33] Ellison, N. B., Steinfield, C., and Lampe, C. The Benefits of Facebook friends: Social Capital and College Students Use of Online Social Network Sites. *Journal of Computer-Mediated Communication 12*, 4 (2007), 1143–1168.

[34] Fortino, V., and Zhao, W. Video Tutorials that Enhance Laboratory Learning. *Frontiers in Education Conference* (2012), 1–2.

[35] GitHub. Five Years (blog post), 2013. `https://github.com/blog/1470-five-years`.

[36] Glaser, B., and Strauss, A. *The Discovery of Grounded Theory: Strategies for Wualitative Research*. Aldine Publishing Company, New York, 1967.

[37] Glaser, B. G., and Strauss, A. L. *Awareness of Dying*. Transaction Publishers, New Jersey, 1966.

[38] Gubrium, J. F., Ed. *The Sage Handbook of Interview Research: The Complexity of the Craft*. Sage Publications, London, 2012.

[39] Gueorguieva, V. Voters, Myspace, and YouTube the Impact of Alternative Communication Channels on the 2006 Election Cycle and Beyond. *Social Science Computer Review 26*, 3 (2008), 288–300.

[40] Hallberg, L. R. The Core Category of Grounded Theory: Making Constant Comparisons. *Intl Journal of Qualitative Studies on Health and Well-being 1*, 3 (2006), 141–148.

[41] Harry, B., Sturges, K. M., and Klingner, J. K. Mapping the Process: An Exemplar of Process and Challenge in Grounded Theory Analysis. *Educational Researcher 34*, 2 (2005), 3–13.

[42] Hemetsberger, A., and Reinhardt, C. Sharing and Creating Knowledge in Open-Source Communities: The Case of KDE. *The Fifth European Conference on Organizational Knowledge, Learning, and Capabilities* (2004), 1–4.

[43] Hollan, J., Hutchins, E., and Kirsh, D. Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research. *ACM Transactions on Computer-Human Interaction 7*, 2 (2000), 174–196.

[44] Jones, M., and Alony, I. Guiding the Use of Grounded Theory in Doctoral Studies, An Example from the Australian Film Industry. *Intl Journal of Doctoral Studies 6* (2011), 95.

[45] Kelle, U. Emergence vs. Forcing of Empirical Data? A Crucial Problem of Grounded Theory Reconsidered. *Historical Social Research/Historische Sozialforschung* (2007), 133–156.

[46] Kocejko, T. The Influence of Multimedia Based E-Learning Techniques for the Capability of Adopting the Knowledge by Senior Students. *14th Intl Conference on Interactive Collaborative Learning* (2011), 237–240.

[47] Kotlarsky, J., and Oshri, I. Social Ties, Knowledge Sharing and Successful Collaboration in Globally Distributed System Development Projects. *European Journal of Information Systems 14*, 1 (2005), 37–48.

[48] Krippendorff, K. *Content Analysis: An Introduction to Its Methodology*. Sage Publications, London, 2012.

[49] Lethbridge, T. C., Singer, J., and Forward, A. How Software Engineers Use Documentation: The State of the Practice. *IEEE Software 20*, 6 (2003), 35–39.

[50] Levy, M. WEB 2.0 Implications on Knowledge Management. *Journal of Knowledge Management 13*, 1 (2009), 120–134.

[51] Lindvall, M., and Rus, I. Knowledge Management in Software Engineering. *IEEE Software 19*, 3 (2002), 26–38.

[52] MacLeod, L. Reputation on Stack Exchange: Tag, You're It! *28th Intl Conference on Advanced Information Networking and Applications Workshops* (2014), 670–674.

[53] MacQueen, K. M., McLellan, E., Kay, K., and Milstein, B. Codebook Development for Team-Based Qualitative Analysis. *Cultural Anthropology Methods 10*, 2 (1998), 31–36.

[54] Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. Design Lessons from the Fastest Q& A Site in the West. *Proc. of the SIGCHI Conference on Human Factors in Computing Systems* (2011), 2857–2866.

[55] McLure Wasko, M., and Faraj, S. It is What One Does: Why People Participate and Help Others in Electronic Communities of Practice. *The Journal of Strategic Information Systems 9*, 2 (2000), 155–173.

[56] Moghaddam, A. Coding Issues in Grounded Theory. *Issues in Educational Research 16*, 1 (2006), 52–66.

[57] Mohorovicic, S. Creation and Use of Screencasts in Higher Education. *35th Intl Convention on Information and Communication Technology, Electronics and Microelectronics* (2012), 1293–1298.

[58] Morgan, G., and Smircich, L. The Case for Qualitative Research. *Academy of Management Review 5*, 4 (1980), 491–500.

[59] Mullamphy, D., Higgins, P., Belward, S., Ward, L. M., et al. To Screencast or Not to Screencast. *Anziam Journal 51* (2010), 446–460.

[60] Nonaka, I. The Knowledge-Creating Company. *Harvard Business Review 69*, 6 (2007).

[61] Oud, J. Guidelines for Effective Online Instruction Using Multimedia Screencasts. *Reference Services Review 37*, 2 (2009), 164–177.

[62] Pace, S. A Grounded Theory of the Flow Experiences of Web Users. *Intl Journal of Human-Computer Studies 60*, 3 (2004), 327–363.

[63] Panahi, S., Watson, J., and Partridge, H. Social Media and Tacit Knowledge Sharing: Developing a Conceptual Model. *World Academy of Science, Engineering and Technology*, 64 (2012), 1095–1102.

[64] Parnin, C., Treude, C., Grammel, L., and Storey, M.-A. Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow. *Georgia Institute of Technology, Tech. Report* (2012).

[65] Pennington, N. Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs. *Cognitive Psychology 19*, 3 (1987), 295–341.

[66] Ploderer, B., Thomas, P., and Howard, S. Being Online, Living Offline: The Influence of Social Ties over the Appropriation of Social Network Sites. *Proc. of the 2008 ACM conference on CSCW* (2008), 333–342.

[67] Quantcast. Quantcast Stack Overflow Statistics, 2014. `https://www.quantcast.com/stackoverflow.com`.

[68] Schwandt, T. A. *The Sage Dictionary of Qualitative Inquiry.* Sage Publications, London, 2007.

[69] Seaman, C. B. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering 25*, 4 (1999), 557–572.

[70] Singer, L., Figueira Filho, F., Cleary, B., Treude, C., Storey, M.-A., and Schneider, K. Mutual Assessment in the Social Programmer Ecosystem: An Empirical Investigation of Developer Profile Aggregators. *Proc. of the 2008 ACM Conference on CSCW* (2013), 103–116.

[71] Singer, L., Figueira Filho, F. M., and Storey, M.-A. D. Software Engineering at the Speed of Light: How Developers Stay Current Using Twitter. *Proc. of the 36th Intl Conference on Software Engineering* (2014), 211–221.

[72] Stemler, S. E. A Comparison of Consensus, Consistency, and Measurement Approaches to Estimating Interrater Reliability. *Practical Assessment, Research & Evaluation 9*, 4 (2004), 66–78.

[73] Storey, M.-a. Theories, Methods and Tools in Program Comprehension: Past, Present and Future. *13th Intl Workshop on Program Comprehension* (2005), 181–191.

[74] Storey, M.-A., Cheng, L.-T., Bull, I., and Rigby, P. Waypointing and Social Tagging to Support Program Navigation. *Proc. of the 20th Conference on CSCW* (2006), 1367–1372.

[75] Storey, M.-A., Singer, L., Cleary, B., Figueira Filho, F., and Zagalsky, A. The R Evolution of Social Media in Software Engineering. *Proc. of the on Future of Software Engineering* (2014), 100–116.

[76] Storey, M.-A., Treude, C., van Deursen, A., and Cheng, L.-T. The Impact of Social Media on Software Engineering Practices and Tools. *Proc. of*

the FSE/SDP Workshop on Future of Software Engineering Research (2010), 359–364.

[77] Strauss, A., and Corbin, J. Grounded Theory Methodology. *Handbook of Qualitative Research* (1994), 273–285.

[78] Suddaby, R. From the Editors: What Grounded Theory is Not. *Academy of Management Journal 49*, 4 (2006), 633–642.

[79] Sugar, W., Brown, A., and Luterbach, K. Examining the Anatomy of a Screencast: Uncovering Common Elements and Instructional Strategies. *The Intl Review of Research in Open and Distance Learning 11*, 3 (2010), 1–20.

[80] Tempelman-Kluit, N. Multimedia Learning Theories and Online Instruction. *College & Research Libraries 67*, 4 (2006), 364–369.

[81] Treude, C., Barzilay, O., and Storey, M.-A. How do Programmers Ask and Answer Questions on the Web? *Proc. of the 33rd Intl Conference on Software Engineering* (2011), 804–807.

[82] Treude, C., Figueira Filho, F., Cleary, B., and Storey, M.-A. Programming in a Socially Networked World: The Evolution of the Social Programmer. *The Future of Collaborative Software Development* (2012), 1–3.

[83] Udell, J. Name that Genre: Screencast, 2004. `http://jonudell.net/udell/2004-11-17-name-that-genre-screencast.html`.

[84] Von Mayrhauser, A., and Vans, A. M. Program Comprehension During Software Maintenance and Evolution. *Computer 28*, 8 (1995), 44–55.

[85] Walenstein, A. Foundations of Cognitive Support: Toward Abstract Patterns of Usefulness. *Interactive Systems: Design, Specification, and Verification* (2002), 133–147.

[86] Wenger, E. Communities of Practice and Social Learning Systems. *Organization 7*, 2 (2000), 225–246.

[87] YouTube. YouTube Press Statistics , 2013. `https://www.youtube.com/yt/press/statistics.html`.