Selection and Presentation Practices for Code Example Summarization

Annie T. T. Ying and Martin P. Robillard
Presented by Tianxiao Deng

Background

1.Code examples are an important source for answering questions about software libraries and applications.

2.Many usage contexts for code examples require them to be distilled to their essence (e.g., when serving as cues to longer documents, or for reminding developers of a previously known idiom.)

3.Programmers search for code examples frequently and extensively. Nearly a third of the respondents in a survey of programmers searched for code examples every day.

4.Code examples are an expected component of formal API documentation[2]

5.On popular forums such as Stack Overflow, 65% of accepted answers contain code examples [3], while unanswered questions often lack code [1]

What makes a code example effective?

- Concise examples also tend to be in highly rated answers on the developer forum Stack Overflow.
- In contrast, longer code examples can be difficult to understand [2] or even be misleading [4], and cause serious presentation problems for summarizing documents.

Need technology to automatically shorten a source code fragment. Unfortunately, no such technology exists.

Related Work

Nasehi et al. investigated the characteristics of code examples in highly rated answers on Stack Overflow [5]. They found that these examples tend to be concise": the examples are **typically less than four lines** and shorter than similar code inside other answers to the same question", “with reduced complexity" and “unnecessary details" left out.

Buse and Weimer studied code examples found in an authoritative source of code examples: the official Java JDK documentation [6].
Their two findings :

1. **markers such as ellipses were employed to indicate an input variable's context-specific value**,
2. **exception handling code was in many JDK examples**

Rodeghero et al.'s recent study specifically looked into whether three types of Abstract Syntax Tree (AST) nodes were important for selecting which part of the code is important for a summary or explanation, by tracking eye movements of participants during a code-to-text summarization task.

Study Set-Up

- **Goal of the study** :To learn code summarization practices and their justification from human participants to inform future development in source code summarization and presentation technology
- **Two research questions**:

  1.Selection: Which parts of the code from an original code fragment should be selected for a summary, and why?

  2. Presentation: How should the code be presented in a summary, and why?

RQ will be answered based on selection practices and presentation practices discussed later.

Study Set-Up

- Recruited **16** participants and asked each of them to shorten **10** code fragments.
- Used think-aloud protocol [7] to instruct the participants to verbalize their thought process.
- In order to estimate differences in personal style, for each code fragment, asked 3 participants to shorten it and the result of which we call **summary**
- **In total ,** 156 summaries on 52 code fragments and 26 hours of screen-recording with synchronized audio.

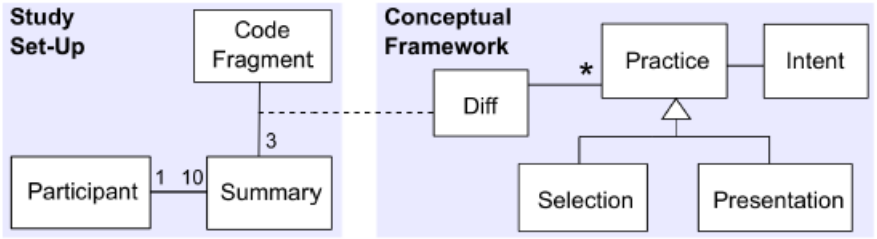


Figure 1: Study Set-Up and Conceptual Framework

Details of Study

- Summarization Task

  1.The participants used a data collection tool designed for this study

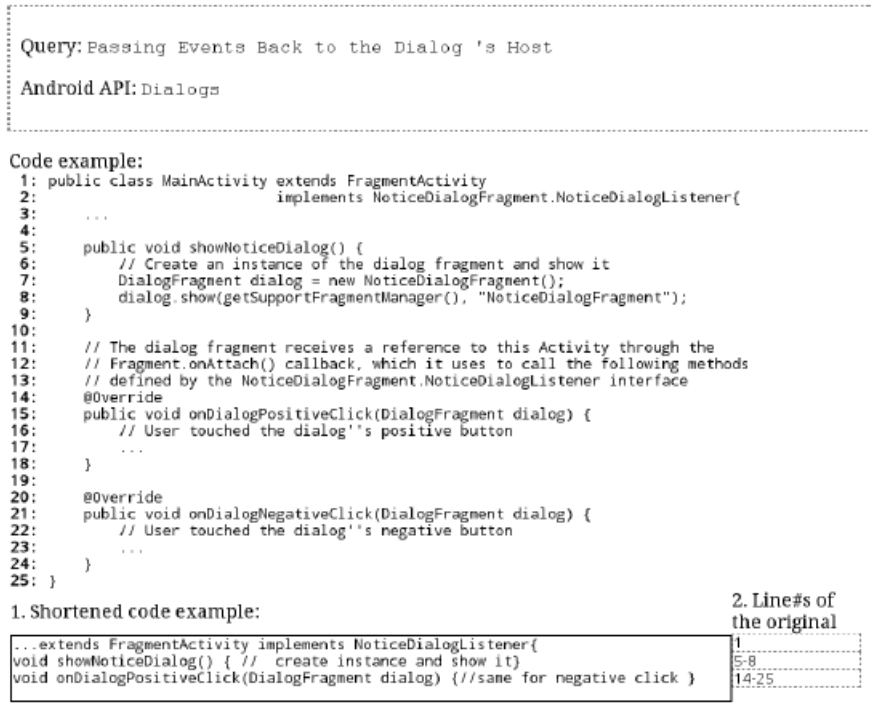


Figure 2: Annotation tool, with a summary by P6

contextual information

original code fragment

fixed-sized text box for writing summary

Details of Study

- Summarization Task

2.The participants verbalized their thought process for the entire duration of their summarization activities. The verbalizations were recorded together with a video of the screen.

3.The study have multiple authors summarizing the same code example so that we could examine the variability among different code summary authors.

4. The summarization task was constrained to limit summaries to three lines.

Details of Study

- Code Fragments

Selecting code fragments has two challenges.

1. To distill a fragment to its essence, participants need a basic idea of what the fragment is about.

2. Code summarization requires a non-trivial level of programming expertise.

Solutions:

Selecting the code fragments from a well-defined corpus of programming documents : The Official Android API Guides.(contains a mix of natural-language text and code fragments). Allow us to draw from the structure of the text surrounding a code example to provide the context and to explicitly scope the expertise required of participants.

Details of Study

- Participants

1. Assigned the 52 fragments to the 16 participants(P1-P16). Twelve participants were assigned 10 fragments and four were assigned 9 fragments. All fragments were summarized by exactly three participants .

2. All participants have one year or more of Java programming experience and have at least looked at the Android API.

Table 1: Participants' Development Experience

Java \ Android	looked at Android API	developed an app	professional
1 year	P3,6,7,8	P4,5,10	
between 1 & 5 yrs	P9,14,15	P1	P11
between 5 &10 yrs		P2	P16
more than 10 yrs			P12,13

Details of Study

- Analysis

1. The study produced two different types of data: shortened source code and the verbalizations of participants. We analyzed this data using a combination of quantitative and qualitative methods.

2. Systematically extract the textual differences between code fragments and the corresponding summaries. And refined the difference into a structured list of **summarization practices.**(two types : ” Selection ” and “ Presentation”)

Threats to Validity

- The corpus of code fragments is limited to 52 fragments in one technology. It is not representative of any defined population of code fragments besides the Android documentation.

- It is possible that not all practices are equally likely to be observed in the 52 fragments. Some useful practices could be ignored.

- The data is collected directly from participants and is influenced by them. The corresponding threat is that a participant with an unusual background or behaving strangely could corrupt the data.

Selection Practices

- Method

All participants selected to including method

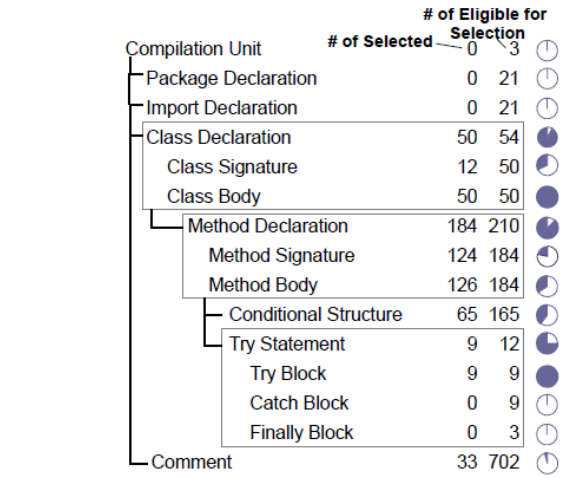
Practice - Including (or Excluding) the Method Signature:

Including method signature

Including method signature but excluding method body

Including both method signature and method body

Excluding method signature



	# of Selected	# of Eligible for Selection
Compilation Unit	0	3
Package Declaration	0	21
Import Declaration	0	21
Class Declaration	50	54
Class Signature	12	50
Class Body	50	50
Method Declaration	184	210
Method Signature	124	184
Method Body	126	184
Conditional Structure	65	165
Try Statement	9	12
Try Block	9	9
Catch Block	0	9
Finally Block	0	3
Comment	33	702

Figure 4: How often a construct was in a summary

Most of participant choose to include both method body and method signature

Selection Practices

- Method [redacted] All participants selected to including method

Practice - Including Overriding Methods

Of the method declarations with an explicit @Override annotation (43 methods), most of the methods (36) were included in a summary by at least one participant.

However only in six fragments , the override annotation itself was kept [redacted]

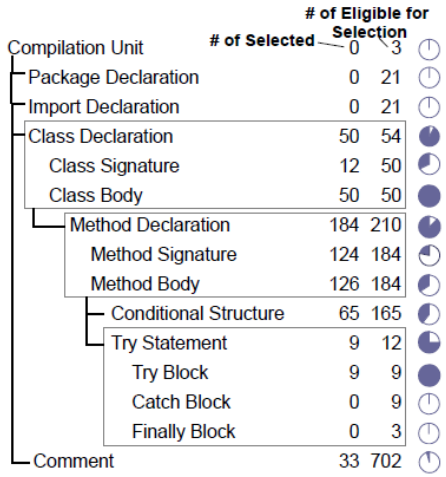


Figure 4: How often a construct was in a summary

Selection Practices

Practice - Excluding Exception Handling Blocks:

None of the exception handling code, enclosed in catch or finally blocks, appeared in a summary.

Practice - Keeping Only One Case in a Parallel Structure:

Some code fragments contained code with multiple cases.
In the case of if or switch statements, more than one third of the instances only had one block selected for a summary.

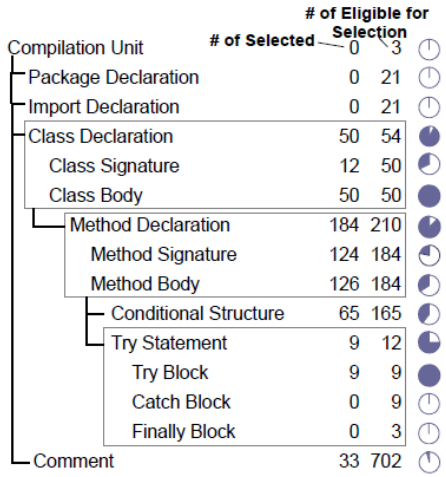


Figure 4: How often a construct was in a summary

Selection Practices

Practice - Based on Query Terms

participants used terms from the query to determine whether a part of the code was relevant enough to include in a summary. Thirteen out of 16 participants explicitly mentioned the importance of the query in the decision of content selection.

Selection Practices

- Practices Considering the Human Reader

Practice - Including Easy-to-Miss Code:

Four participants mentioned including easy-to-miss parts of the code in the summary.

Practice - Accounting for Programming Expertise:

Seven participants justified not including parts of the code that were too obvious to the reader.

Practice - Using the Query to Infer Expertise:

Participants used the query to infer the level of expertise on the API of the query poser, and then excluded the part of the API deemed obvious.

Presentation Practices

- Trimming a Line When Needed

Ten participants performed transformations for the purpose of trimming a line, such as shortening variable names or removing a type qualifier. 

Practice – Shortening Identifier: 

Eight participants did so in 29 (56%) code fragments. By (1) using acronyms (2) shortening words in an identifier (3) dropping words or paraphrasing

Practice – Eliding Type Information: 

Practice - Shortening API Names: 

Presentation Practices

- Compressing a Large Amount of Code

Twelve participants employed more complex abstraction and aggregation practices that greatly reduced the code from its original size.

Practice – Shortening Multiple Statements:

Ten participants shortened multiple statements including the whole method body. The use of comments versus ellipses was split almost evenly

Practice – Shortening Method Declarations:

Seven participants aggregated whole method declarations by replacing the whole declaration with comments or with ellipses.

```
1 /* implement SensorEventListener @override ↩
     onAccuracyChanged(), onSensorchanged() */
2 @override onCreate(), onAccuracyChanged(), onResume()↩
     , onPause() onSensorChanged(){...}}
3 // remember to override all inherited methods ↩
     appropriately
```

Figure 5: All three summaries on the same example contained a comment listing overriding methods

Presentation Practices

- Compressing a Large Amount of Code

  Practice - Shortening Control Structures:

  Eight participants shortened control structures.

```
1 while(cur.moveToNext()) {...}}
2 if (checked)... else...}
3 if (resultCode == Activity.RESULT_OK && requestCode ↩
      ==  PICK_CONTACT_REQUEST) { //code for activity }
```

Figure 6: Sample of summarized control structures

- Truncating Code

  Twelve participants performed truncation

  Practice - Eliminating a Parameter:

  By replacing a parameter with ellipses or simply eliminating a parameter

  Practice - Truncating a Signature:

  Changes involved Java keywords (such as public or static), identifier names, or the whole signature replaced by a comment.

Presentation Practices

- Formatting Code for Readability

Practice - Indenting Code: 

Practice - Keeping Lines as Separate: 

All participants treated at least one summary with all separate lines.

Table 2: Evidence of the presentation practices

Presentation practices	#Participants (out of 16)	#Fragments (out of 52)	# Instances
Trimming a Line When Needed	10	P2,4,6,8,10,11,13,14,15,16	33	95
Shortening Identifiers	8	P2,4,6,8,10,11,15,16	29	72
Shortening API Names	4	P6,10,11,15	5	7
Eliding Type Information	10	P2,4,6,8,10,11,13,14,15,16	9	16
Compressing a Large Amount of Code	13	P2,3,4,5,6,7,8,9,10,11,14,15,16	28	46
Shortening Multiple Statements	10	P3,4,6,7,8,9,10,11,15,16	15	51
Shortening Method Declarations	7	P4,6,8,10,11,14,16	10	11
Shortening Control Structures	8	P2,3,5,6,8,10,11,16	12	14
Truncating Code	12	P1,2,4,5,6,8,9,10,12,13,15,16	28	63
Eliminating a Parameter	9	P1,4,5,6,8,9,10,15,16	16	28
Truncating a Signature	9	P2,4,5,8,10,12,13,16	18	35
Formatting for Readability	16	all	52	140
Indenting	8	P1,2,4,7,8,9,13,14	20	27
Treating Lines as Separate	15	all except P10	52	135
Improving Code	9	P1,2,4,6,8,10,11,12,16	9	35
Fowler's Refactorings	4	P1,2,8,11	4	5
Generalization	4	P2,4,8,16	5	8
Clarification	6	P6,8,10,11,12,16	14	22

#Summaries (out of 156)

Conclusion

This study elicited selection and presentation practices observed from 156 concise code representations obtained from 16 participants. The selection practices revealed the importance of the human reader, that participants targeted summaries to the expertise level inferred from the query. All 16 participants employed practices to modify the content, mostly with the intent to make it more concise but also make it more compilable, readable, and understandable. The practices directly inform the design and the generation of concise source code representations.

Discussion

1. Accounting for expertise information to determine which content should be included can be a complement of existing code example search engines . Existing measures to quantify expertise include the use of commit logs and interaction history. What other measures can we apply?

2.What are problems that you usually have about code example?

3.Do you have any idea to improve quality of code example?

References

1. M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A.Schneider. Answering questions about unanswered questions of stack overow. In Proceedings of the Working Conference on Mining Software Repositories,Challenge Track, pages 97-100, 2013.
2. M. Robillard and R. DeLine. A eld study of API learning obstacles. Empirical Software Engineering,16(6):703{732, 2011.
3. S. Subramanian and R. Holmes. Making sense of onlinecode snippets. In Proceedings of the Working Conference on Mining Software Repositories, Challenge Track, pages 85-88, 2013.
4. E. Cutrell and Z. Guan. What are you looking for? An eye-tracking study of information usage in web search. In Proceedings of the Conference on Human Factors in Computing Systems, pages 407-416, 2007.
5. S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example? A study of programming Q&A in StackOverow. In Proceedings of the International Conference on Software Maintenance, pages 25-34, 2012.
6. R. Buse and W. Weimer. Synthesizing API usage examples. In Proceedings of the International Conference on Software Engineering, pages 782-792, 2012.
7. C. Lewis and J. Rieman. Task-Centered User Interface Design: A Practical Introduction, chapter 5: Testing The Design With Users. Self-published, 1993. http://grouplab.cpsc.ucalgary.ca/saul/hci topics/tcsd- book/contents.html.