Institute of Software Engineering
Empirical Software Engineering Group

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Validating the Threats to Validity in Program Comprehension Experiments

Marvin Muñoz Barón

Course of Study:	Softwaretechnik
Examiner:	Dr. Daniel Graziotin
Supervisor:	Marvin Wyrich, M.Sc.
Commenced:	November 16, 2021
Completed:	May 16, 2022

Abstract

BACKGROUND: Designing empirical software engineering methodologies presupposes the existence of evidence to base actions on empirical fact. Understanding what factors may influence experiment results, how to select appropriate samples and study subjects, and how to correctly apply statistical methods is a prerequisite for developing a methodology based on established research. This is especially true in a field such as program comprehension, where hundreds of different contextual factors can alter the validity of the obtained results. Here, researchers often have intuitions about what might threaten the validity of their studies but do not have the evidence to support their claims.

OBJECTIVE: This study examines the threats to validity in program comprehension experiments to collect evidence of their existence, to understand the context and nature in which they occur, and to ultimately assist researchers in designing controlled experiments with high validity.

METHODS: First, we conduct a systematic review surveying existing program comprehension experiments and summarizing what threats to validity they report. We then follow up on the three most commonly cited threats, performing small-scale systematic reviews and evaluating the collected evidence using an evidence profile to investigate their influence as a threat.

RESULTS: We found that only 31 out of 409 (8 %) individual threat mentions were reported with supporting evidence. Furthermore, for the three most common threats, programming experience, program length and comprehension measures, we found that contextual factors such as how measurements are made, the individual characteristics of the population sample, and what concrete tasks are employed all change the way a threat impacts the results of a study.

CONCLUSION: Threats to validity are highly context-dependent and as such must be controlled in different ways. Researchers should use existing evidence to inform their decision-making and explicitly address both why a threat poses a danger and how they controlled it in the context of their study. To this end, we need structured guidelines for reporting threats to validity and public knowledge bases that contain threats, evidence, and mitigation techniques for program comprehension experiments.

Contents

List of Figures

List of Tables

Acronyms

EBSE evidence-based software engineering. 23

EEG electroencephalography. 16

EiPE explain in plain English. 15

fMRI functional magnetic resonance imaging. 16

LOC lines of code. 34

NIRS near infra-red spectroscopy. 16

Chapter 1

Introduction

In human-centered experiments such as those on program comprehension, dozens if not hundreds of different factors can act as potential confounders for the results of a study [1]. Reporting potentially confounding factors and other threats to the validity of scientific research has long been seen as good practice, and the number of program comprehension studies that do so continues to grow [2]. While this indicates an awareness of researchers of threats to validity in their studies, program comprehension studies employ starkly differing methodologies, leading to difficulties when comparing their results [3]. Researchers often have an intuition on what could influence their experiment results, but lack evidence-based resources to support their claims.

This study conducts an investigation into the threats to validity in program comprehension to address these ambiguities. To this end, we first analyze the current state of reporting in scientific literature. Using a systematic review, we summarize which and how researchers discuss threats to validity in 95 papers reporting on primary experiments of program comprehension. Moreover, we append an evidence-gathering phase to this review, in which we further examine the three most commonly cited threats to validity: programming experience, program length, and comprehension measures. During this process, we conduct separate small-scale systematic reviews to identify existing evidence on the validity of each threat. We then evaluate each piece of evidence against an evidence profile and render a final verdict for each case.

Structure

The rest of this report is organized as follows:

Chapter 2: Overview of the background and related works.
Chapter 3: Explanation of the methods used to answer our research questions, including systematic review, evidence collection, and the evidence profile.
Chapter 4: Results and discussion of our systematic review of threats to validity in program comprehension experiments.
Chapter 5: Results and discussion of the evidence collected for the three most common threats.
Chapter 6: Discussion of the threats to validity of this study.
Chapter 7: Summary, conclusion and suggestions for future work.

Chapter 2

Background

This work draws on the extensive body of research in software engineering and related fields. In this section, we provide a brief overview of the relevant topics and explain the basic concepts that drive our methodology.

2.1 Program Comprehension

In computer science, the ability to read and understand the source code of computer programs is often the most basic task one first learns. This process, commonly referred to as **program comprehension**, is the act of deriving the functionality of a program by interacting with its code artifacts. Shaft et al. differentiate between the processes of top-down and bottom-up program comprehension [4]. The former refers to applying existing domain knowledge to generate understanding from macro-level code artifacts such as entire programs, components, and documents. The latter refers to the detailed reading of micro-level artifacts such as code snippets, lines of code, and individual program constructs. O'Brien et al. [5] further refine this model by decomposing top-down comprehension into two distinct types of understanding. They distinguish between expectation-based comprehension, which describes the construction of understanding with pre-existing expectations of a code's meaning and inference-based comprehension, in which a programmer infers the meaning of code from well-known implementations. Programmers use both top-down and bottom-up comprehension strategies to varying degrees, depending on their familiarity with the codebase and the stage of development [6].

Considerable research effort has gone into qualitative and quantitative experimentation to better understand the underlying processes and parameters that govern the human understanding of source code. A schematic representation of such an experiment is shown in figure 2.1. A common approach is to ask participants to solve a task that requires them to understand a particular program in order to complete it successfully. Their performance on this task is then gauged by measures of comprehension, which represent proxies for understandability. These proxy measures can be viewed as operationalizations of the difficulty of understanding in this specific comprehension process. Using these measures, researchers can compare various factors that influence comprehension and evaluate approaches in terms of their ability to help programmers understand code.

Due to their complexity, controlled comprehension experiments can vary considerably in terms of the types of comprehension tasks and measures employed [3, 7]. However, there are some tasks and measures that have gained acceptance in the research community and are generally believed to capture the code comprehension performance of participants. One example is comprehension questions, where a participant reads a code snippet and then answers questions about its functionality or purpose [8–11]. These can range from multiple-choice and explain in plain English (EiPE) tasks to tracing questions, in which a participant must trace an input through the program and compute its value at various stages of program execution. Another common task is to recall certain aspects

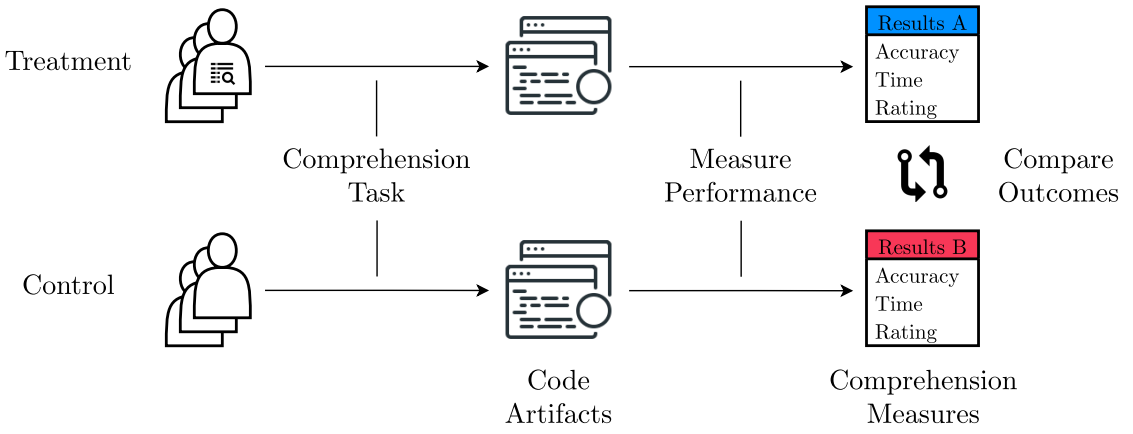


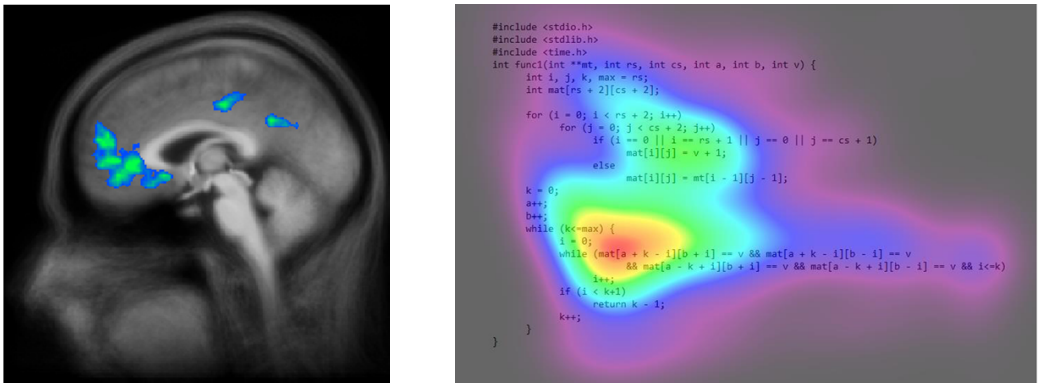
Figure 2.1: A schematic depiction showing the artifacts, actions, and actors involved in a comprehension experiment.

of the code, such as identifier names or entire lines of code, testing a participant's memorization ability. An example of this are cloze tests [12], where participants are asked to fill in areas of the code that have intentionally been removed [13–15]. In other cases, participants are asked to give their subjective opinion about the difficulty of a comprehension task in the form of a rating [11, 14–16]. By exploiting the assumption that a programmer must understand a program to modify it, some experiments involve more complex maintenance activities such as identifying [17, 18] and fixing bugs [19, 20]. Their efficacy in completing these activities is then used as an indicator of their levels of understanding of the program.

The performance of participants in comprehension tasks can also be measured in a variety of different ways. Task accuracy is measured by the correctness of a participant's response [8, 10, 11, 14]. This can be, for example, the number of correctly answered questions or a subjective evaluation of the responses by an expert. Different measures of time may also be taken during the comprehension process. Depending on the specific task, an experimenter might capture the time it takes a participant to submit their response or how long it takes them to find the correct solution [8, 11, 14, 15].

Moreover, cross-discipline approaches introduce the possibility of gathering physiological measures. In eye-tracking studies, a visual capturing device is placed in front of a participant as they are reading code to record their eye movements. Eye-trackers can provide information on which code areas participants are looking at, where their gaze lingers, and when their pupils are dilated [21–24]. Conversely, functional magnetic resonance imaging (fMRI), electroencephalography (EEG), and near infra-red spectroscopy (NIRS) technology allows researchers to directly monitor brain activity during program comprehension [22, 23, 25, 26]. Figure 2.2 shows examples of how physiological measures can be used to visualize different aspects of the comprehension process.

In addition to the choice of treatment, comprehension task, and understandability measures, other confounding factors must be considered during experimentation [1]. Previous research has investigated the influence of personal factors such as experience [29–31] and gender [32], code-related factors such as identifier names [18, 33, 34], comments [15, 33, 35], programming paradigms [36–38], and environmental factors such as presentation [39–41] on program comprehension. Since



(a) fMRI **(b)** Heat map generated from gaze data

Figure 2.2: (a) An fMRI showing areas of the brain with significant brain deactivation [27] and (b) a heat map showing code areas with high visual attention during program comprehension [28].

the process of understanding is vastly complex, many different factors may influence the behavior and performance of participants in these experiments. To ensure reliable results, researchers must consider potential threats to validity when designing their experiments.

2.2 Experiment Validity

Rosenthal et al. describe measurement validity as "the degree to which the measures are appropriate or meaningful in the way they claim to be" [42]. To ensure validity, a comprehensive experimental strategy must be developed that avoids common pitfalls and explicitly addresses the different facets of validity. The validity of experiments can be further divided into specific types that require different strategies to mitigate any factors that could threaten them [43]:

- Ensuring **internal validity** means explicitly controlling potential confounding factors to better reason about the causality of the relationship between the independent treatment and observed effect of the study.
- **External validity** refers to the degree to which results obtained in the experiment can be generalized to other populations or settings. To ensure high external validity, an experimenter must select a sample that adequately represents the population and design an experiment that accurately reflects the particular action as it occurs outside of the study.
- **Construct validity** refers to whether the constructs measured in the experiment do in fact represent the real-world concepts under study. For example, in program comprehension, one might question whether the time it takes to deliver a response is an accurate indicator of the concept of understandability.
- **Conclusion validity** refers to whether the conclusion reached through the methods employed in the study is correct. To ensure high conclusion validity, researchers must thoroughly design their methodology to produce the same results under independent replication.

A common practice in scientific literature is to report the limitations of a study in a “threats to validity” section. Kitchenham et al., in their guidelines for conducting systematic reviews, recommend the inclusion of a discussion of the validity of the presented evidence [44]. Therein, researchers explicitly point out the parts of their methodology that could jeopardize the study’s validity and the factors that have to be considered when assessing its results. These sections help the reader to better gauge the scope of application of the study by keeping its limitations in mind.

2.3 Evidence-based Software Engineering

Evidence-based practice has a long history in medicine and healthcare, where administering a treatment without evidence of its efficacy is considered malpractice. Drugs and vaccines go through a long series of trials to prove their safety and effectiveness before they are ever used in practice. In software engineering, attempts have been made to establish similar approaches to help researchers find accurate answers to research questions and to help practitioners make informed decisions in their everyday work [44]. While primary experiments and case studies form the basis of scientific evidence, meta-studies are commonly considered the gold standard. In systematic reviews and meta-analyses, researchers survey existing studies with the goal of summarizing, discussing, and combining findings to achieve a better understanding of the field as a whole. Such meta-studies are exceedingly complex and require both meticulous effort and sophisticated strategies [45]. However, evidence-based practice may also be adapted for use in an industrial context, for example, through techniques such as rapid reviews [46] and evidence profiles [47].

2.4 Related Works

The study presented in this report represents a systematic review and collection of evidence on the threats to validity in program comprehension experiments. In this section, we briefly describe related studies and compare them to our approach.

In their systematic mapping study, Zhou et al. [45] analyze the current state of threats to validity reported in systematic reviews in the software engineering domain. First, they identify existing reviews in literature using a systematic search strategy and then extract key information from the relevant papers. They divide their findings into the threats themselves, their consequences, and potential mitigation techniques. Their results show that, when counted, biases in data extraction and study selection as well as inappropriate or incomplete search terms were reported most frequently, resulting in the misclassification of publications, missing or excluded relevant primary studies, and overall data inaccuracies. Based on their analysis, they recommend several mitigation strategies for conducting systematic reviews. These recommendations include establishing a detailed search protocol in the planning phase of the review, using a strategy that involves well-defined criteria for the inclusion and exclusion of papers, employing a combination of manual and automatic searches and snowballing, and using multiple sources to search for literature. They also emphasize the importance of independent review of both the search protocol and the filtering decisions.

Biffl et al. [48] created a knowledge base of threats to validity in software engineering experiments to assist researchers in planning their studies. The content of the knowledge base was compiled from the results of the systematic review conducted by Neto et al. [49] and general threats discussed

in standard software engineering textbooks. They found that only a small set of threats are reported by many studies, while a majority the threats are too specific to be generalized independent of the particular research area. They conclude by stating that there is a need for an overview of threats to validity as they are reported in the specific areas of software engineering research.

Siegmund et al. [1] surveyed the literature published in several software engineering publications to obtain information about confounding parameters in program understanding. The main insights they gained were that each paper reported only a small subset of all confounding factors, most of which were reported in the sections on experimental design and threats to validity, and that researchers used different mitigation techniques to address the same factors. They also cataloged the individual threats reported and counted the frequency of their occurrence. The most frequently reported participant characteristics were programming experience and familiarity with the tools and study object. The most cited experimental factors were potential learning effects and the size and programming language of the study objects. They recommend that other researchers include the identification of confounding parameters in their experimental design and explicitly report the relevant parameters and how they controlled them.

In our approach, similar to the studies discussed above, we examine the state of the art of reported threats to validity in software engineering. We mirror some of the methodological strategies employed by Zhou, Biffl, and Siegmund et al. in summarizing existing threats, such as using a systematic search protocol, analyzing the frequency of reported threats, and specifying both threats and mitigation techniques. Additionally, we incorporate Zhou et al.'s recommendations for conducting systematic reviews in our research design. Differences arise from the focus area of the study, where, like Siegmund et al., we investigate studies of program comprehension rather than systematic reviews or software engineering studies in general. Furthermore, we append our review process by including a period of evidence collection, where we conduct small-scale reviews to gather evidence on the most common threats. A detailed explanation of the entire methodology used can be found in chapter 3.

Chapter 3

Methods

In this section, we describe the main research objectives and the methodology used to achieve them. The goal of this study is to provide researchers with clarity on what factors threaten the validity of program comprehension studies and to identify areas of insufficient evidence, providing clear starting points for further research. To this end, we formulated the following main research questions.

RQ1: What threats to validity are reported in studies of program comprehension?
Understanding the current state of reporting on the threats to validity in scientific literature can provide valuable insight into what researchers consider to be the most pressing issues in their studies. Based on this knowledge, they may carefully adjust their study design plans to incorporate mitigation techniques for controlling proven confounding factors.

RQ2: Is there evidence to support the most frequently reported threats to validity in studies of program comprehension?
While knowing which threats are most frequently reported on might provide some indication as to what the center of discussion is in program comprehension, it does not necessarily provide clarity about their empiricism. Therefore, it is necessary to investigate whether the factors that are intuitively considered to be the most important also have scientific evidence to support them. In chapter 5 we further refine this question with respect to the three most commonly reported threats.

RQ3: How can systematic evidence collection help researchers design studies with high validity?
The approach to evidence collection used in this work is relatively novel. While its constituent tools such as systematic review and evidence profiling have been used in the past, the constellation used to validate threats to validity in our methodology has yet to be scrutinized. This research question presents a critical look at how the approach covered in this work has helped to achieve the goal of understanding validity threats in program comprehension. In addition, it is intended to provide guidance to other researchers in adopting similar methods in other research areas.

To answer **RQ1**, we first systematically analyze and summarize the threats to validity reported in existing studies of program comprehension as described in section 3.1. Based on these results, we prioritize the most frequently cited threats and conduct further systematic reviews, collecting evidence to answer **RQ2** explained in section 3.2. Finally, we assess said evidence through the use of an evidence profile, providing a final overview of the state of evidence for the three most commonly reported threats described in section 3.3.

3.1 Systematic Review

The first step in our methodology is to summarize the reported validity threats through systematic review. We follow the guidelines of Kitchenham for conducting systematic reviews in software engineering [44].

Search Source

When conducting a review, relevant papers must first be collected through a systematic search of relevant literature databases and subsequent filtering according to explicit inclusion and exclusion criteria. In the case of this work, the list of relevant primary papers had already been identified through a previously conducted systematic search. This search identified 95 papers reporting on empirical studies of bottom-up program comprehension with human participants, published in a peer-reviewed journal, conference, or workshop before 2020. Moreover, one of the authors of this systematic search is the supervisor of this master thesis and provided the list of identified papers to be used for this review.

Review Strategy

The list of primary papers is then analyzed individually, categorizing and summarizing each threat to validity reported in the full text. We follow a similar summarization strategy to Zhou et al. [45] who studied common threats to validity of literature reviews in software engineering. In particular, we adopt a thematic synthesis [50] approach to identify the individual threats to validity and if given, the mitigation techniques mentioned by the study. Relevant text areas are extracted from the paper and inductive coding [51] is used to find appropriate codes to describe the passage. In inductive coding, codes are formulated during the review process as the corresponding concepts become evident. Throughout the coding process, these codes are refined and reapplied, improving their quality through iteration. Upon the completion of code assignment, the threat codes are then categorized and composed into high-level themes. Papers may already contain evidence of the threat in the form of references to other studies, which are also documented. It is likely that the studies will have varying degrees of completeness with respect to each of these aspects, as they may elaborate more or less on each individual threat to validity. The goal of this work is to highlight where information is lacking and further research is needed.

In summary, the review process consists of the following individual steps:

1. Extraction of relevant text passages on threats to validity, mitigation techniques, and evidence.
2. Inductive coding where a describing code is assigned to each threat.
3. Categorization of threat codes and composition into higher-order themes.

Table 3.1 illustrates this approach by showing extracted information for a single threat.

Threat Code	Novices perform worse in comprehension tasks reducing external validity
Mitigation	Select a sample including both novices and experts, ...
Evidence	Mustermann et al., ...
Category	Programming Experience
Theme	Personal factors

Table 3.1: Example of the relevant factors extracted during the review process.

Prioritization

In the previous step, each threat was assigned a describing code and those referring to the same concept were grouped into categories. These threat categories most closely resemble a traditional threat to validity or confounding factor, as threat codes that differ slightly in wording or context but still refer to the same concept are grouped together. However, this loss of nuance from grouping the individual codes into categories is only necessary during the prioritization of threats. The full threat codes were still considered when formulating the refined research questions, the search terms, in the evaluation, and in the final discussion. The most frequently mentioned threat categories are prioritized and further analyzed in the evidence collection described in section 3.2.

3.2 Evidence Collection

At this point, we have a list of threats, how frequently they were reported, and in some cases, a pool of starting evidence. In other words: We found that researchers have an intuition about a threat and its potential consequences for the validity of a study. The next step is to see if the existing body of research supports their intuitions. Collecting evidence to support or refute a particular theory is one of the core tenets of evidence-based software engineering (EBSE) [44]. Due to the large number of different threats, we cannot conduct a systematic review to find evidence for each of them. Instead, we focus on the most frequently cited threats. Specifically, we use a systematic approach to find evidence in the form of studies that examine a threat to validity and its role in program comprehension experiments. The steps described in this section and in section 3.3 are repeated individually for each of the common threats.

Search Protocol

The search protocol used in this study uses several different sources to search for potentially relevant papers. We describe each of these sources and list the filtering criteria applied to the literature found within them. Furthermore, we describe how we used snowballing [52] as a technique to further extend the search. Backward snowballing in this context means including the reference list of a paper, while forward snowballing means including papers that cite the paper in question.

(A) Primary Papers

The 95 papers with primary research on program comprehension may already examine the threat in question as part of their research. Consequently, they are analyzed and evaluated as potential evidence. Backward snowballing for this set of papers is not required because potential evidence in their references has already been previously collected (see (B)). Forward snowballing is unlikely to yield relevant results, as investigating the threat in question was not the main focus of these papers.

(B) Evidence Reported in the Primary Papers

As part of the review of threats to validity reported in the 95 primary papers, we identified and documented all evidence cited to support the assertions made about each threat. This pool of starting evidence is analyzed and filtered in the same manner as the primary papers. This evidence plays a similar role as the papers in (A) in that they do not necessarily focus on the threat as their

primary object of research which is why forward snowballing is unlikely to yield relevant results. However, backward snowballing is performed here as they may refer to similar evidence when comparing their results with other works.

(C) Evidence Found through the Title Search

To further enrich the dataset with research from sources independent of the primary papers, we also conduct small-scale searches. This reduces the bias created by limiting our analysis to the primary papers and allows us to consider literature published between 2020 and 2022. We focus our search on studies that mention the particular threat to validity in their title. We used the search strings shown in figure 3.1 in Google Scholar[1] to find evidence for the three most common threats. Both backward and forward snowballing is valuable here, as the studies found in the title search are likely to have the threat in question as the main subject of their research.

1. Programming Experience

```
allintitle: (experience OR novice OR expert) (code OR software OR program)
(understandability OR comprehension OR comprehensibility OR readability OR
analyzability OR "cognitive load")
```

2. Program Length

```
allintitle: (size OR length OR short OR long OR LOC OR "lines of code") (code OR
software OR program) (understandability OR comprehension OR comprehensibility OR
readability OR analyzability OR "cognitive load")
```

3. Comprehension Measures

```
allintitle: (measure OR measures OR measurement) (code OR software OR program)
(understandability OR comprehension OR comprehensibility OR readability OR
analyzability OR "cognitive load")
```

Figure 3.1: Search strings used in the evidence collection.

(D) Evidence Found through Snowballing

Snowballing is used to put further emphasis on relevant papers by including their reference list as an additional search source. As described in the previous steps, backward snowballing is used on (B) and (C) and forward snowballing is used on (C).

All papers contained in (A) to (D) are filtered according to the criteria listed in table 3.2.

3.3 Evidence Profile

After all available evidence of a particular threat has been collected, that evidence must be evaluated. To this end, we employ the evidence profile proposed by Wohlin [47]. This profile is a model for evaluating evidence based on criminal law. Each piece of evidence is judged individually and classified into different types depending on how strong the evidence is. In addition, the profile

[1] https://scholar.google.com/

Inclusion criteria

- Reports on a primary experiment or case study measuring program comprehension.
- Analyzes the threat in question as part of their study.

Exclusion criteria

- Full text of the paper is not accessible.
- The paper is not available in English.
- Published outside of peer-reviewed journals or conference proceedings.

Table 3.2: Inclusion and exclusion criteria.

distinguishes between positive and negative evidence, with positive evidence supporting the theory in question and negative evidence contradicting it. It is important to stress that "bad evidence", meaning evidence of low quality, is not synonymous with contradictory evidence. Contradictory evidence can be of high quality but oppose the notion that a threat has an influence on the results of an experiment. Thus, a score close to zero indicates the strength of a piece of evidence is low, while a negative or positive score gives an indication of the outcome of the study. The evidence types are shown in table 3.3.

Score	Type	Description
+5/-5	Strong evidence	Studies that focused on the threat in question as the main subject of their investigation or conducted an in-depth analysis of the threat as part of their overall approach and show significant results. Systematic reviews that examine the threat in question and provide a conclusion.
+4/-4	Evidence	Studies that did not have the threat as their main focus but still included it in their analysis. These studies may have more uncontrolled confounding factors which is why they should be considered separate from stronger evidence.
+3/-3	Circumstantial evidence	Similar to evidence, studies that are considered circumstantial evidence do not have the threat as the main focus of their study. Furthermore, they show additional methodological shortcomings that reduce the reliability of the study and decrease its strength as evidence in our evaluation.
+2/-2	Third-party claim	Studies that make claims about the threat in question but only provide a slim level of empirical backing for said claim. They may defer to other sources of information or give general impressions about the influence of the threat in their study, but provide no dedicated statistical analysis of their own to support their claims.
+1/-1	First- or second-party claim	Studies that make a claim about the threat in question but do not provide any empirical backing. This could, for example, be references to "common knowledge" or speculation.

Table 3.3: Types of evidence according to the evidence profile. Higher values mean stronger evidence. Negative values contradict the threat, positive values support it.

Note that due to the flexible nature of the evidence profile, the descriptions do not match the ones provided by Wohlin word-for-word. Wohlin emphasizes that the evidence profile should be adapted to the context in which it is used. For example, aspects such as *vested interest* and the *aging of evidence* do not play a major role when collecting evidence on confounding parameters unless the threat pertains to a specific technology or approach that may influence experiment results. On the other hand, the methodological rigor captured in the *quality of evidence* as well as the *relevance of the evidence* is of utmost importance and is therefore emphasized in the evidence types.

Placement of a study in a particular level is based on its adherence to the given type description as well as on the previously mentioned quality aspects. Therefore, a study may be placed in a lower category if it has significant shortcomings regarding any of the quality aspects. Because the evaluation of evidence is a largely subjective process, it is conducted independently by two different researchers to reduce bias. Once both researchers have scored a piece of evidence, a final score is selected by comparing the scores of both researchers. In case of divergent results, they discuss their disagreements and agree on a singular score. The score of each study and the motivation behind its placement are transparently documented and can be used to paint an overall picture of the evidence landscape for each threat. Using these techniques, we provide a conclusive recommendation to researchers. In some cases, there might be insufficient, contradictory, or no evidence. Here, suggestions for further research are made based on the information gathered.

Chapter 4

Systematic Review: Validity Threats in Program Comprehension Experiments

In this section, we present the results of the systematic review of threats to validity in program comprehension experiments. First, we present the results and then discuss them with our own interpretations.

RQ1: What threats to validity are reported in studies of program comprehension?
Every controlled experiment has some unique elements in its research design that distinguish it from other studies or replications. However, some of these factors could affect the validity of the study results. Explicitly addressing and documenting such threats helps experimenters design experiments that produce reliable results and helps other researchers interpret and compare results. To understand the current state of the art regarding threats to validity in program comprehension experiments, we analyzed the full text of 95 papers reporting on them and summarized the reported threats using coding techniques.

4.1 Results

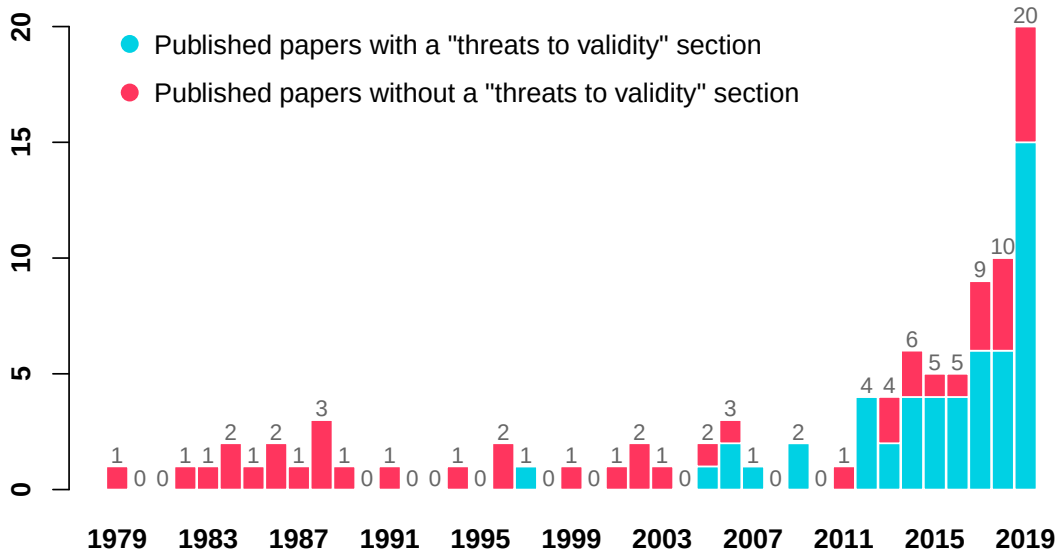


Figure 4.1: Number of primary papers published per year.

We first present how threats to validity were reported in general. Among the 95 papers, 81 (85 %) mentioned at least one threat to validity, with 45 (47 %) reporting them in a dedicated section and 33 (35 %) differentiating between different types of validity, such as internal or external validity. We also identified trends with regards to the publications over time. Figure 4.1 shows how many of the analyzed studies were published each year. In addition, studies that included a section on threats to validity are highlighted in blue, and those that did not are highlighted in red. The largest concentration of studies is found in 2012 to 2019, with a total of 63 of the 95 studies (66 %), while only covering 7 years (18 %) of the entire 40-year span.

Theme and Category	Count
Theme: Code Snippets	**112**
Length	26
Complexity	16
Code Selection	13
Programming Language	9
Synthetic Samples	9
Familiarity	6
Theme: Participant Factors	**101**
Programming Experience	44
Number of Participants	16
Programming Skills	10
Theme: Experimentation	**89**
Learning Effect	18
Lab Experiment	11
Fatigue	9
Code Presentation	7
Cheating	6
Theme: Measurement	**67**
Comprehension Measures	22
Eye-Tracking	20
Instrumentation	9
Theme: Comprehension Tasks	**21**
Type of Comprehension Task	7
Task Difficulty	6
Theme: Data Analysis	**14**
Statistics	10
Theme: Other	**5**
Total	**409**

Table 4.1: Number of threat mentions per category and theme.

Threat Code	Count
Missing diversity in length of code snippets leads to limited generalizability	23
Missing diversity in participant's programming experience leads to limited generalizability	22
Learning effect when performing comprehension tasks	16
Diversity in participant's programming experience confound treatment effects	13
Missing diversity in complexity of code snippets leads to limited generalizability	11
Fatigue when performing comprehension tasks	9
Inadequate application of statistical methods	9
Researchers may introduce bias when selecting code snippets	8
Inadequate measures of comprehension	8
Low number of participants	8

Table 4.2: Most common threat codes.

Next, we present the result of the analysis of the content of the reported threats themselves. In total, we found 409 individual threat mentions that were assigned 215 unique threat codes. When multiple threat mentions referred to exactly the same concept, they were assigned the same threat code. Out of the 409 threat mentions, only 31 (8 %) were reported with supporting evidence, while 198 (48 %), or nearly half, included an explanation of a possible mitigation technique. Moreover, these 31 references to supporting evidence were found in 20 out of 95 studies (21 %).

The most common threat codes with over 10 mentions can be found in table 4.2. These threat codes were the smallest coding unit we used and summarize the text passage where the threat occurs. Some of the threats referred to the same concept, but differed in nuance and therefore received different codes. After grouping similar threat codes into threat categories, we were left with 81 unique threats. Table 4.1 highlights the threat categories with more than five reported threat mentions and shows the themes to which they were assigned. Themes are presented with the total number of threat mentions in all categories, not exclusively those highlighted in the table. The three most common threat categories were programming experience, program length, and comprehension measures. Overall, most threats were related to the characteristics of the code snippets, the individual factors of the participants, or general threats in experimentation.

4.2 Discussion

There is a clear bias in our data towards more recent studies, as there was an increase not only in the total number of papers analyzed per year but also in the absolute number of threats reported per paper. However, this bias is difficult to avoid, as the number of software engineering studies in general increases with each passing year. We were able to replicate to some extent the findings of Shroter et al. [2] who showed that more recent studies of program comprehension more frequently report threats to validity. In our case, before 1997, there was not a single study that included a section on threats to validity, and by the year 2012, most studies had such a section. However, it should be noted that the maximum number of studies published per year was less than 3 until 2012, which may skew the data somewhat.

As part of our prioritization, we counted how often each threat to validity occurred. Similarly, Siegmund et al. [1] examined the frequency with which various confounding factors were reported in experiments on program comprehension. For most factors, our results coincide, with the most common factors being programming experience, learning effects, the size and language of the study object, familiarity with the study object, and general experimental factors. There are also differences in some other areas. Threats such as the number of participants, comprehension measures, and problems related to eye-tracking, which were reported particularly often in our study, are absent in the Siegmund et al. study. Some of the factors from our study can be mapped to related factors in their study, such as instrumentation to technical problems in eye-tracking, study-object coverage to the number of participants, and mono-operation and mono-method bias to comprehension measures, but all of these factors generally occurred less frequently. This may be due to different coding techniques, as we group some threats under a different umbrella term or split some terms into different parts.

The most pressing discovery we made was that only 8 % of the total threat mentions were supported with any kind of evidence. This indicates that there is a common practice of using speculation when describing threats to validity, with researchers having intuitions or knowledge about what

might threaten the validity of their studies but not citing literature to corroborate their claims. The question that immediately follows is whether this practice is due to a lack of existing evidence for these threats, or whether such evidence exists but remains unused. We attempt to answer this question by conducting an evidence search for the three most common threats in section 3.2.

RQ1: Main Findings

- Most program comprehension studies report threats to validity
- Few support validity threats with corresponding evidence
- Most recent studies have a dedicated threats to validity section
- The three most commonly reported threats were **programming experience**, **program length**, and **comprehension measures**

Chapter 5

Evidence Collection: The Three Most Common Threats to Validity

Based on the results of our systematic review, we selected the three most frequently reported threats and collected evidence on their validity. These threats are programming experience, found in section 5.1, program length in section 5.2, and comprehension measures in section 5.3. The full evidence lists including justifications for the placement of studies in the different evidence categories can be found in appendix A.

5.1 Threat 1: Programming Experience

When designing experiments, one should select a sample that represents the population of interest. Furthermore, each participant will inevitably have some individual factors that could potentially influence the results of the experiment. To adequately control these confounding factors, we must first understand if and how their influence occurs. In this analysis, we examine the effect a participant's programming experience has on their code understanding in comprehension experiments.

RQ2.1: Does the programming experience of developers affect their ability to comprehend code?
Programming experience refers to the time spent learning, writing, and working with program code. Usually, this is characterized by the number of years a programmer has been writing code or working professionally in the software industry. Furthermore, a person with little programming experience is considered a novice and someone with a lot of experience is considered an expert. We collect evidence in the form of studies that measured the programming experience of participants and used statistical tests to determine whether it had a significant influence on program comprehension.

Results

Table 5.1 shows how many papers were found in each step of the evidence collection and how many were excluded because they did not meet the inclusion criteria. Overall, most evidence was found in the primary papers, followed by snowballing. We excluded 11 of 12 (92 %) documents cited in the primary papers as they did not meet the filtering criteria. The reasons for exclusion varied and are described in more detail in the discussion.

After filtering, 60 papers remained and were evaluated as potential pieces of evidence using the evidence profile. In this evaluation, 11 additional papers were discarded as they did not meet the criteria to be considered evidence. The final evidence profile is shown in figure 5.1. The result largely favors programming experience as a confounder for program comprehension. In

Source	Analyzed	Excluded	Final
(A) Primary Papers	95	-52	43
(B) Evidence Reported in Primary Papers	13	-12	1
(C) Evidence Found through Title Search	54	-52	2
(D) Evidence Found through Snowballing	276	-262	14
Evidence Profile	60	-11	49
Total	**438**	**-389**	**49**

Table 5.1: Overview of the evidence analyzed and filtered in each step of first evidence collection.

total, 37 (76 %) pieces of evidence were rated as supporting evidence and 12 (24 %) were rated as contradicting evidence. Furthermore, there were 11 pieces of strong supporting evidence and no strong contradicting evidence.

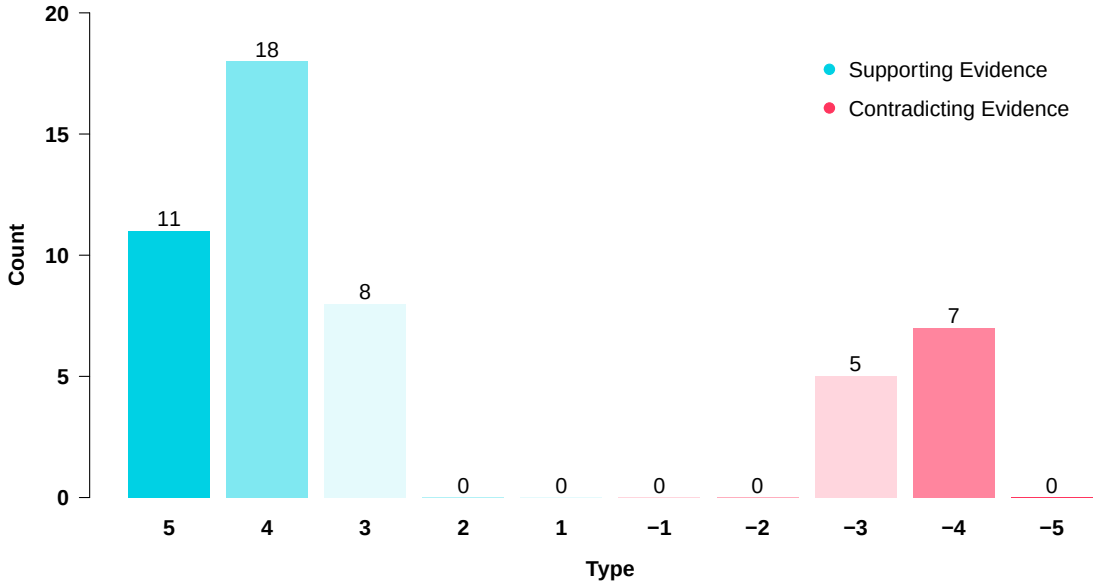


Figure 5.1: Evidence Profile 1: Does the programming experience of developers affect their ability to comprehend code?

Discussion

Looking at the filtering process, we find some unexpected discoveries. First, the studies used in the primary papers to support claims about threats to validity were, with one exception, almost entirely dismissed as evidence. They mostly were excluded because they either did not relate to program comprehension and tended to focus on experts versus novices in software engineering research in general, or were related to comprehension, but not to programming experience. In some cases, the evidence was solely claim a on validity without conducting experiments and in others it was not relevant at all.

Second, snowballing proved to be quite valuable. Even though only three papers were used for snowballing, the number of relevant pieces of evidence that were found was 14. However, to find those 14 pieces of evidence, 274 additional studies had to be evaluated. This illustrates that while snowballing is valuable, it can quickly snowball into an exponentially increasing number of papers to be filtered.

Overall, the result of the evidence profile confirms that, in many cases, programming experience does indeed influence the comprehension performance of programmers. In multiple cases, experienced programmers showed different comprehension behavior when compared to novices [53, 54] and this difference could be measured in their performance [55–57]. In contrast, we also found multiple pieces of credible evidence contradicting those assertions. Seven individual studies that showed no methodological shortcomings reached the opposite conclusion with respect to our research question. The question that now arises is why it seems that programming experience has an impact on source code understanding in some cases and not in others. We attempt to find the answer to this question in the context and experimental parameters of each of these studies. For one of these factors, Siegmund et al. [58] found that depending on how programming experience is measured and operationalized, its predictive power varies. This hypothesis is further supported by the fact that some of the contradicting evidence still found correlations with very specific types of experience measures such as self-estimated Java knowledge [27] or correlations for only specific comprehension measures such as the number of eye fixations [59]. In other cases, the range of programming experience was quite limited, for example only including students as participants [27, 60]. It is possible that different results would have been obtained if the range of experience had been more diverse. In most cases, however, the experience analysis was only a small subset of their overall results and thus the sample selection was not intended to maximize differences in experience.

This information leads us to the general conclusion that programming experience plays a role in the comprehension of programmers. However, the extent and nature of this influence depends on various contextual factors. These include factors such as the sample selection, experiment design, and how experience is measured in the first place. Researchers should explicitly analyze and state the context factors in their comprehension experiment and control for them appropriately.

RQ2.1: Main Findings

- Many studies investigate the effect of programming experience
- The majority of studies show a positive outcome, but some contradict the hypothesis
- Researchers should assess the experience of participants and use it as a controlling factor
- Depending on the study context, novices and experts might display different comprehension behavior

5.2 Threat 2: Program Length

Regardless of the specific focus of a study, examining program comprehension means, at its core, examining program artifacts. Participants read source code that has certain characteristics that affect their perception of the code. Researchers must be aware of which characteristics need to be controlled to ensure reliable experimental results. In this analysis, we examine how the length of a program affects program comprehension.

RQ2.2: Do treatment effects observed in short code snippets also appear in longer snippets or larger code bases?
Program length or size refers to the number of syntactic statements in a program, usually measured in lines of code (LOC). We gather evidence in the form of studies that measure the length of a program and examine its impact on program comprehension.

Results

Table 5.2 shows how many papers were found in each step of the evidence collection and how many were excluded because they did not meet the inclusion criteria. The only evidence was found in the primary papers. All evidence cited in the primary papers was excluded as it did not meet the filtering criteria. Moreover, as no relevant papers were found in (B) and (C), no snowballing was performed.

Source	Analyzed	Excluded	Final
(A) Primary Papers	95	-78	17
(B) Evidence Reported in Primary Papers	3	-3	0
(C) Evidence Found through Title Search	29	-29	0
(D) Evidence Found through Snowballing	0	0	0
Evidence Profile	17	-4	13
Total	**127**	**-114**	**13**

Table 5.2: Overview of the evidence analyzed and filtered in each step of second evidence collection.

After filtering, 17 papers remained and were evaluated as potential pieces of evidence using the evidence profile. In this evaluation, 4 more papers were discarded as they did not meet the criteria to be considered evidence. The final evidence profile is shown in figure 5.2. We found conflicting results with regards to the influence of program length on program comprehension. In total, 6 (46 %) pieces of evidence were rated as supporting evidence and 7 (54 %) were rated as contradicting evidence. Furthermore, there were 2 pieces of strong supporting evidence and no strong contradicting evidence.

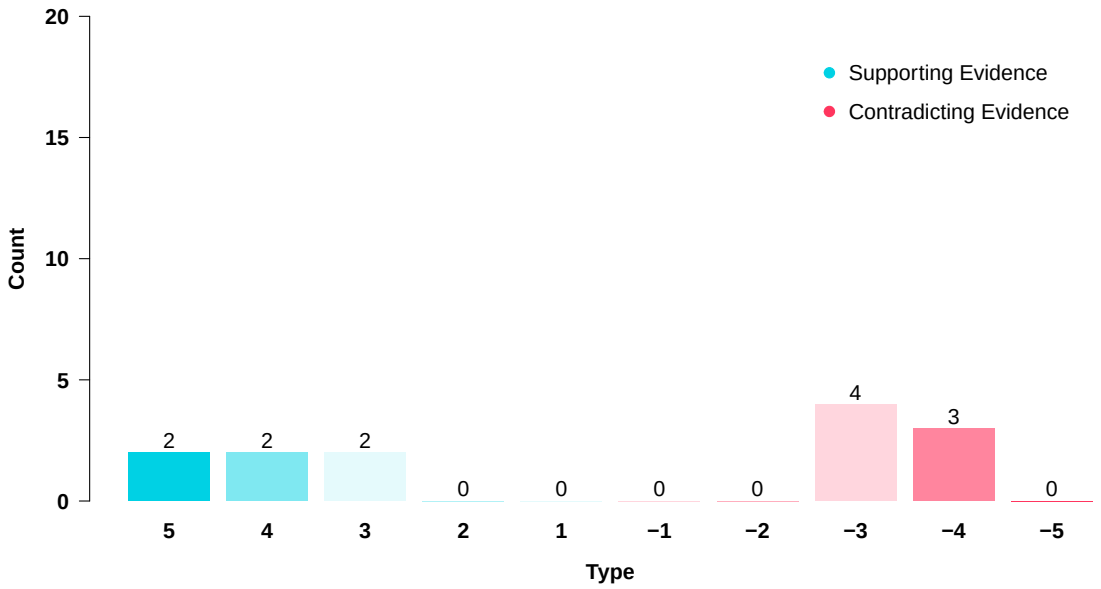


Figure 5.2: Evidence Profile 2: Do treatment effects observed in short code snippets also appear in longer snippets or larger code bases?

Discussion

During the evidence filtering, it stood out that no evidence was found in both the evidence citations of the primary papers and the title search. For the primary paper evidence, this is consistent with the pattern uncovered in the first evidence collection, where these studies did not adequately address the threat in question. What was unexpected, however, were the results of the title search. Here, not a single study met the criteria for inclusion. There are a few possible explanations for this. On the one hand, it could simply be a shortcoming of our search strategy, meaning that there is relevant evidence but we did not search for it correctly. While this is always a possibility, we followed the same search strategy, down to the same search string pattern as in the first and third evidence collection, both of which found relevant evidence in their respective title searches. However, another likely explanation is that there is only a small amount of strong evidence, which is why we did not find any in the title search. By its very nature, the title search finds mainly studies that focus on the threat in question, since it is mentioned in the title. Consequently, if there is close to no strong evidence in existence, then the title search will come up empty. This is somewhat supported by the fact that even in the 95 primary papers, only 2 presented strong evidence of this threat.

Looking at the results of the evidence profile, we find a more contentious result than in the first evidence collection. There was more than three times more evidence in the first evidence collection and there is no clear indication to the validity of the threat one way or the other. While there is one more piece of contradicting evidence, on the other hand, there are two strong pieces of evidence supporting the threat. Ribeiro et al. [61] also discuss similarly conflicting evidence they found on program length in their study. Even after conducting a follow-up study to resolve these conflicts, they were unable to reach a clear conclusion about the influence of program size on program comprehension. They found that all participants preferred snippets with more lines of code for

comprehensibility, but only novices found them to be more readable. Experts instead preferred short code snippets in terms of readability. They also mention that differences in the procedures and tools used to measure lines of code may affect the comparability of results from different studies.

Overall, our main finding is that there is a need for more research investigating program length as a confounding factor. Even though it is the second most cited threat to validity in program comprehension experiments, there are very few studies examining its influence. For the little evidence that we did find, we come to the same conclusion as in the first evidence search. There is some evidence that program length has an influence on program comprehension and should be considered a threat to validity. However, this influence is context-dependent, meaning that it is subject to factors that vary for each individual study and must therefore be controlled differently in accordance with those factors.

RQ2.2: Main Findings

- Few studies explicitly investigate program length as a confounding factor
- Existing evidence on program length as a confounding factor for program comprehension is conflicting
- The impact of program length as a threat to validity is context-dependent

5.3 Threat 3: Comprehension Measures

Comprehension experiments often involve participants completing some sort of task that requires them to understand a code artifact. However, there are a variety of different methods that researchers use to measure the performance of participants on these tasks. These comprehension measures serve as proxies for the concept of comprehension. In this analysis, we examine the validity of common comprehension measures by investigating whether they are associated with distinct aspects of comprehension.

RQ2.3: Are commonly used comprehension measures not correlated and associated with distinct aspects of program comprehension?
The level of comprehension of a code artifact exhibited by a participant is often measured by the time it takes them to complete a task, the accuracy or correctness of that completion, and their subjective rating of the difficulty of the task. Furthermore, physiological measures may be taken to study the effect the task completion has on the body of a participant. We gather evidence in the form of comparative studies that analyze correlations between these commonly used comprehension measures.

Results

Table 5.3 shows how many papers were found in each step of the evidence collection and how many were excluded because they did not meet the inclusion criteria. Overall, most evidence was found in the primary papers, with slightly less evidence found in both the title search and through snowballing. All evidence cited in the primary papers was excluded as it did not meet the filtering criteria.

Source	Analyzed	Excluded	Final
(A) Primary Papers	95	-88	7
(B) Evidence Reported in Primary Papers	3	-3	0
(C) Evidence Found through Title Search	53	-51	2
(D) Evidence Found through Snowballing	134	-131	3
Evidence Profile	12	-5	7
Total	**285**	**-278**	**7**

Table 5.3: Overview of the evidence analyzed and filtered in each step of third evidence collection.

After filtering, 12 papers remained and were evaluated as potential pieces of evidence using the evidence profile. In this evaluation, 5 more papers were discarded as they did not meet the criteria to be considered evidence. The final evidence profile is shown in figure 5.3. Most evidence supported that the commonly used measures measure distinct aspects of program understanding and are not correlated. But overall, only a small amount of evidence was found. In total, 5 (71 %) pieces of evidence were rated as supporting evidence and 2 (29 %) were rated as contradicting evidence. Furthermore, there were 3 pieces of strong supporting evidence and no strong contradicting evidence.

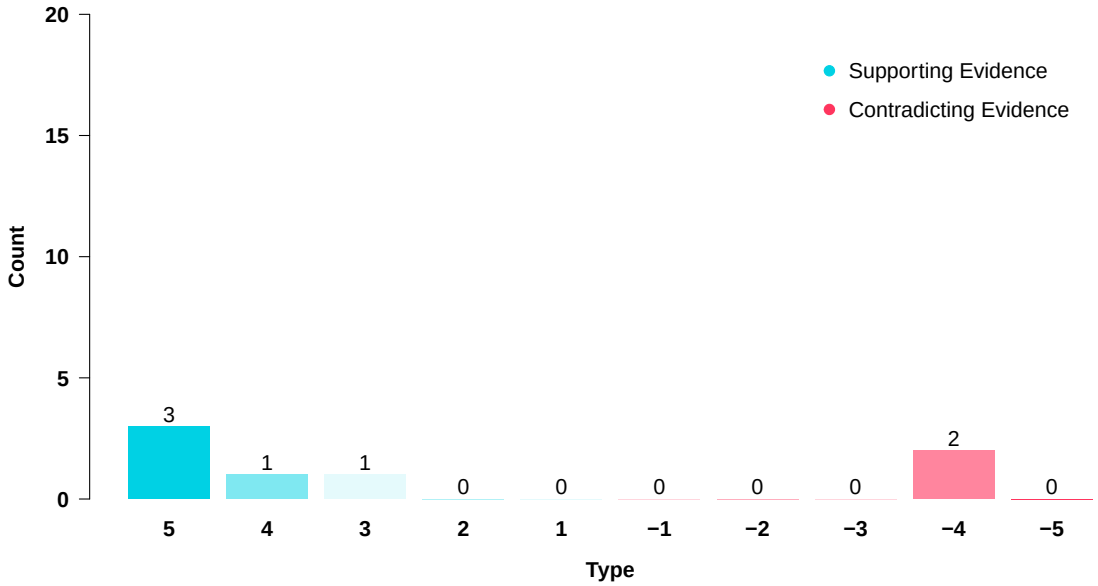


Figure 5.3: Evidence Profile 3: Are commonly used comprehension measures not correlated and associated with distinct aspects of program comprehension?

Discussion

As with the first two searches, we found no evidence in the citations of the primary papers. We also again observed the same pattern of the evidence reported in the primary papers not adequately supporting the claims made about the threats to validity.

The results of the evidence profile mostly support the notion that the commonly reported measures are not correlated. However, to gain a better understanding of each variable, they are first examined separately. Looking at each piece of evidence individually, we find some differences in their methodology and which measures they compared. Both pieces of contradicting evidence found correlations between the time and accuracy of comprehension tasks and both were published before the year 1990. In one case, the code understanding of non-programmers was explored using comprehension questions, and in another, the results of cloze tests were compared with those of multiple-choice questions [13, 62]. One of the supporting pieces of evidence also reported a small but insignificant correlation between time and accuracy [63]. However, three supporting pieces of evidence found no correlations between the time and accuracy of comprehension task performances, suggesting that they measure different effects, aspects or dimensions of comprehension task difficulty [32, 64, 65]. Furthermore, the two studies that compared physiological measures with other measures found no correlation with subjective ratings [63] and time and accuracy of comprehension tasks [24, 63]. The significance of this finding becomes even clearer when one considers that of the 95 primary papers, more than one third (37) used only a single measure to assess participants' comprehension performance. Depending on the study context, this may cause them to miss aspects of code understanding that would have been uncovered if they had analyzed more than one measure. This result also awards some merit to the 47 primary studies that measured multiple variables but decided to analyze each variable separately.

If different proxy measures are associated with different aspects of program comprehension, this poses further difficulties for meta-studies of comprehension experiments. For example, two studies might obtain different results because they measured comprehension performance in different ways and not due to other intrinsic factors. For now, blindly mix- and matching studies with different types of measures should be avoided until we better understand why they appear to measure distinct concepts.

Similar to the first two collections of evidence, we find that the impact of the threat to validity depends on which measures are used and that even when the same measures are used, other contextual factors can alter the impact of the threat on experimental outcomes. Again, this reinforces our suggestion that researchers must control threats to validity within the context of their study, adapting procedures, measures, and artifacts to the individual characteristics of the study.

RQ2.3: Main Findings

- Most commonly used comprehension measures are not correlated
- Researchers might miss distinct aspects of comprehension if they only employ a single comprehension measure

5.4 Discussion

In this section, we compare and contextualize the results of all three evidence collections.

RQ2: Is there evidence to support the most frequently reported threats to validity in studies of program comprehension?
In all three evidence searches, we found both supporting and contradicting evidence, seemingly leading to a divergent result. There was no easy or obvious yes or no answer that gave us a clear indication of whether or not a threat posed a danger to study results. Therefore, in all three cases, we investigated whether any common patterns are able to explain this divergence. We found that depending on how a confounding factor was measured, what the sample population was, and how the experiment was designed in general all can change the way a threat impacts the result of a study. All in all, the main finding of the evidence collections can be summarized as: The influence of a threat to validity differs for each individual study and must as such be interpreted in accordance with the contextual factors surrounding it.

While this conclusion might sound obvious at first, we often find generic statements about a potential threat in primary studies. For example, a study might solely mention that they used used students in their experiment without further elaboration. Describing an issue this way, however, does not clarify why that choice is a threat within the context of their study. This issue is problematic regardless of whether references are used to back up assertions, as generic threats are used in place of more nuanced discussions that take the study-specific context factors into account. Moreover, evidence can be used as an additional layer to further support these explanations.

Beside our main finding we also made some other minor observations. First, we found that there was less evidence for less common threats to validity. This is in line with what would be expected intuitively, if less researchers deem that a threat poses a danger to a study's validity then corollary, less will investigate whether that assumption is true or false. Meta-studies such as this one can highlight which threats are less frequently examined and give directions for further experimentation. Additionally, we observed that throughout the entire evidence collection process, not a single study was classified as a -5, which would represent strong contradicting evidence. While we don't have a conclusive reasoning behind this, it is in line with the general tendency of researchers to not publish negative results. Obtaining negative results can be disheartening, but publishing them is nonetheless extremely important and provides value to the scientific community [66–68].

RQ2: Main Findings

- The impact of a threat to validity is highly context-dependent
- Researchers should discuss why a threat occurs in the context of their study
- Evidence can help researchers understand in which context a threat could be an issue

5.5 Designing for Validity Using Evidence-Based Methods

In this section, we briefly discuss the usage of evidence when designing controlled experiments.

RQ3: How can systematic evidence collection help researchers design studies with high validity?
Evidence-based methods can be valuable both in practice and research. The biggest barriers standing between researchers and the implementation of these methods is the large effort involved in conducting systematic reviews to find evidence and difficulties in interpreting their results. When designing experiments, it is unrealistic to conduct a systematic review to collect evidence for each of the dozens of potential threats. Rather, researchers may look to existing research summarizing evidence and then contextualize it by comparing them with the characteristics of their own study. This way, evidence is used both during the design stages of an experiment and when interpreting and discussing its results.

On the other hand, one may inquire about the nature and role of the threats to validity section in scientific literature itself. While we investigated and reported on different descriptive aspects about the threats reported in existing works, no effort was made towards uncovering why researchers chose to include specific threats and how they reported them. Furthermore, this research does not give any guidance with regards to how a paper author ought to write a threats to validity section. While the general inclination presented by this work is towards backing up reported threats with evidence wherever possible, this is not necessarily a sentiment shared by the general scientific community. A threats to validity section may also be a place where researchers should be able to speculate without concrete evidence and then point out these shortcomings as directions for future research. By conducting interviews with prominent authors of scientific papers, a follow-up work may gain clarity on how the research community sees the threats to validity section and may establish firmer guidelines for reporting validity threats.

Chapter 6

Threats to Validity

In this section, we present potential threats to the validity of the results obtained in this study.

Internal Validity As with every systematic search, the selected search terms, sources, and inclusion criteria will influence the coverage of relevant papers. To mitigate this bias, we used multiple different sources and different techniques to acquire papers, such as both searching literature databases and snowballing.

One issue with using an evidence profile, where the rating of evidence is done through qualitative evaluation of humans is the possibility of introducing bias. To mitigate this threat, two researchers independently rated the evidence and then compared their results to reach an agreement. In the systematic review, this bias nevertheless remains, as the coding and categorization process was only done by one person.

External Validity One issue threatening external validity lies within the source of papers for the systematic review of threats to validity. As the list of papers was a result of a previous search, its search parameters were not adjusted to the goals of this work. More specifically, the search was focused on a subset of program comprehension, namely controlled experiments on bottom-up comprehension. This means that the list of the most common threats may not reflect the prioritization of program comprehension experiments in general. In the evidence collection, this shortcoming is mitigated as we added additional search sources.

Another limitation with regards to the second part of the study is that we were only able to analyze three out of dozens of different threats to validity. It is possible that repeating the same methods for the remaining threats will uncover new idiosyncrasies of program comprehension. As such, the results obtained in this work should be generalized to other threats with care.

Construct Validity The concepts analyzed in this work are rather vague, as the concrete research questions we analyzed in the evidence collection were derived after multiple levels of abstraction. What constitutes a threat to validity and whether our descriptions accurately describe them is subject to discussion. The accuracy in this case is influenced by the accuracy of the codes used to summarize them and the individual decisions made when distinguishing them as different concepts. These individual differences can be observed when comparing the list of threats to the results of past reviews [1]. However, there is no easy way to tell if either result is more valid.

Chapter 7

Conclusion

In this work, we presented an investigation into the threats to validity in program comprehension experiments. First, we analyzed the state of the art by systematically reviewing validity threats in 95 primary papers. We found that while most studies reported threats to validity, few supported them with corresponding evidence. Furthermore, only in one case did the supporting evidence meet the inclusion criteria of our evidence search. Next, we identified the most commonly reported threats to validity and searched for evidence supporting or contradicting them. Our evidence collection yielded mostly supporting and some contradictory evidence. Looking closer into the collected evidence and after comparing our results with other meta-studies we come to the conclusion that threats to validity are highly context-dependent. Even the threats that are intuitively expected to affect program comprehension, such as programming experience, depend on how they are measured, the sample population, comprehension tasks, and other context factors. Therefore, we need to take all the individual characteristics of a study into account when assessing potential threats to validity and design methodologies that use evidence as a basis for implementing appropriate mitigation techniques.

Future work may expand on the methodology proposed in this study by investigating additional threats to validity and informing researchers with evidence-based methods. While we have examined three common threats, there are many more that still require validation. Even when looking at just three of the most common threats, we found a trend that less evidence existed for less popular threats uncovering which might indicate even bigger gaps in evidence for those threats that have yet to be analyzed. Moreover, the approach presented in this work may be applied to other domains to investigate how threats to validity affect other types of experiments in software engineering.

Our findings suggest that researchers should both explicitly report how a validity threat affects their studies within its context and address how they controlled for the threat. This goal can be supported by establishing structured guidelines for reporting threats to validity and informing researchers on how they can incorporate evidence into their validity assessments. Furthermore, we need more knowledge documentation on what threats exist, the evidence supporting them, the context in which they occur, and which mitigation techniques can be used to address them. Previous works laid the groundwork in this endeavor by documenting threats in software engineering studies in a knowledge base and providing guidelines for controlling confounding factors in program comprehension experiments, respectively [1, 48]. These works can be further extended by incorporating evidence and documenting threats to validity in a common database.

All in all, we find that threats to validity can be extremely complex. Assessing which contextual factors to consider, how to measure them, and in what ways they should be mitigated can be difficult to navigate. By relying on evidence-based methods, researchers can make these assessments with a foundation in scientific precedent. After all, a threat is credible only if there is substantial evidence of a reasonable danger to the subject in question.

Bibliography

[1] J. Siegmund, J. Schumann, "Confounding parameters on program comprehension: A literature survey," *Empirical Software Engineering*, vol. 20, no. 4, pp. 1159–1192, 2015, ISSN: 1382-3256. DOI: 10.1007/s10664-014-9318-8 (cit. on pp. 13, 16, 19, 29, 43, 45).

[2] I. Schroter, J. Kruger, J. Siegmund, T. Leich, "Comprehending studies on program comprehension," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, IEEE, 2017, pp. 308–311, ISBN: 978-1-5386-0535-6. DOI: 10.1109/ICPC.2017.9 (cit. on pp. 13, 29).

[3] M. Muñoz Barón, M. Wyrich, S. Wagner, "An empirical validation of cognitive complexity as a measure of source code understandability," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, New York, NY, USA: ACM, 10052020, pp. 1–12, ISBN: 9781450375801. DOI: 10.1145/3382494.3410636 (cit. on pp. 13, 15).

[4] T. M. Shaft, I. Vessey, "Research report—the relevance of application domain knowledge: The case of computer program comprehension," *Information Systems Research*, vol. 6, no. 3, pp. 286–299, 1995. DOI: 10.1287/isre.6.3.286 (cit. on p. 15).

[5] M. P. O'Brien, J. Buckley, T. M. Shaft, "Expectation-based, inference-based, and bottom-up software comprehension," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 16, no. 6, pp. 427–447, 2004 (cit. on p. 15).

[6] C. L. CORRITORE, S. Wiedenbeck, "An exploratory study of program comprehension strategies of procedural and object-oriented programmers," *International Journal of Human-Computer Studies*, vol. 54, no. 1, pp. 1–23, 2001 (cit. on p. 15).

[7] J. Siegmund, C. Kästner, S. Apel, A. Brechmann, G. Saake, "Experience from measuring program comprehension - toward a general framework," *1617-5468*, 2013, ISSN: 1617-5468. [Online]. Available: https://dl.gi.de/handle/20.500.12116/17708 (cit. on p. 15).

[8] J. J. Dolado, M. Harman, M. C. Otero, L. Hu, "An empirical investigation of the influence of a type of side effects on program comprehension," *IEEE Transactions on Software Engineering*, vol. 29, no. 7, pp. 665–670, 2003, ISSN: 0098-5589. DOI: 10.1109/TSE.2003.1214329 (cit. on pp. 15, 16, 57).

[9] N. Kasto, J. Whalley, "Measuring the difficulty of code comprehension tasks using software metrics," in *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136*, ser. ACE '13, AUS: Australian Computer Society, Inc, 2013, pp. 59–65, ISBN: 9781921770210 (cit. on pp. 15, 61).

[10] D. Gopstein, J. Iannacone, Y. Yan, L. DeLong, Y. Zhuang, M. K.-C. Yeh, J. Cappos, "Understanding misunderstandings in source code," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, E. Bodden, W. Schäfer, A. van Deursen, A. Zisman, Eds., New York, NY, USA: ACM, 2017, pp. 129–139, ISBN: 9781450351058. DOI: 10.1145/3106237.3106264 (cit. on pp. 15, 16, 59).

[11] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, R. Oliveto, "Automatically assessing code understandability," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 595–613, 2021, ISSN: 0098-5589. DOI: 10.1109/TSE.2019.2901468 (cit. on pp. 15, 16, 57, 61).

[12] W. L. Taylor, ""cloze procedure": A new tool for measuring readability," *Journalism Quarterly*, vol. 30, no. 4, pp. 415–433, 1953 (cit. on p. 16).

[13] W. E. Hall, S. H. Zweben, "The cloze procedure and software comprehensibility measurement," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 5, pp. 608–623, 1986 (cit. on pp. 16, 38, 63).

[14] M. Crosby, J. Scholtz, S. Wiedenbeck, "The roles beacons play in comprehension for novice and expert programmers," 2002 (cit. on pp. 16, 60).

[15] J. Borstler, B. Paech, "The role of method chains and comments in software readability and comprehension—an experiment," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 886–898, 2016, ISSN: 0098-5589. DOI: 10.1109/TSE.2016.2527791 (cit. on pp. 16, 59).

[16] R. P. L. Buse, W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, 2010, ISSN: 0098-5589. DOI: 10.1109/TSE.2009.70 (cit. on pp. 16, 62).

[17] J. Castelhano, I. C. Duarte, C. Ferreira, J. Duraes, H. Madeira, M. Castelo-Branco, "The role of the insula in intuitive expert bug detection in computer code: An fmri study," *Brain imaging and behavior*, vol. 13, no. 3, pp. 623–637, 2019. DOI: 10.1007/s11682-018-9885-1 (cit. on p. 16).

[18] J. C. Hofmeister, J. Siegmund, D. V. Holt, "Shorter identifier names take longer to comprehend," *Empirical Software Engineering*, vol. 24, no. 1, pp. 417–443, 2019, ISSN: 1382-3256. DOI: 10.1007/s10664-018-9621-x (cit. on pp. 16, 58).

[19] G. Beniamini, S. Gingichashvili, A. K. Orbach, D. G. Feitelson, "Meaningful identifier names: The case of single-letter variables," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, IEEE, 2017, pp. 45–54, ISBN: 978-1-5386-0535-6. DOI: 10.1109/ICPC.2017.18 (cit. on p. 16).

[20] J. L. Whalley, R. Lister, E. Thompson, T. Clear, P. Robbins, P. K. Ajith Kumar, C. Prasad, "An australasian study of reading and comprehension skills in novice programmers, using the bloom and solo taxonomies," *Conferences in Research and Practice in Information Technology Series*, vol. 52, pp. 243–252, 2006 (cit. on p. 16).

[21] B. Sharif, M. Falcone, J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications - ETRA '12*, C. H. Morimoto, H. Istance, J. B. Mulligan, P. Qvarfordt, S. N. Spencer, Eds., New York, New York, USA: ACM Press, 2012, p. 381, ISBN: 9781450312219. DOI: 10.1145/2168556.2168642 (cit. on pp. 16, 59).

[22] N. Peitek, J. Siegmund, C. Parnin, S. Apel, J. C. Hofmeister, A. Brechmann, "Simultaneous measurement of program comprehension with fmri and eye tracking," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, M. Oivo, D. Méndez, A. Mockus, Eds., New York, NY, USA: ACM, 2018, pp. 1–10, ISBN: 9781450358231. DOI: 10.1145/3239235.3240495 (cit. on p. 16).

[23] T. Ishida, H. Uwano, "Synchronized analysis of eye movement and eeg during program comprehension," in *2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP)*, IEEE, 2019, pp. 26–32, ISBN: 978-1-7281-2243-4. DOI: 10.1109/EMIP.2019.00012 (cit. on p. 16).

[24] S. Fakhoury, D. Roy, Y. Ma, V. Arnaoudova, O. Adesope, "Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2140–2178, 2020, ISSN: 1382-3256. DOI: 10.1007/s10664-019-09751-4 (cit. on pp. 16, 38, 63).

[25] T. Nakagawa, Y. Kamei, H. Uwano, A. Monden, K. Matsumoto, D. M. German, "Quantifying programmers' mental workload during program comprehension based on cerebral blood flow measurement: A controlled experiment," in *Companion Proceedings of the 36th International Conference on Software Engineering*, P. Jalote, L. Briand, A. van der Hoek, Eds., New York, NY, USA: ACM, 2014, pp. 448–451, ISBN: 9781450327688. DOI: 10.1145/2591062.2591098 (cit. on p. 16).

[26] A. Duraisingam, R. Palaniappan, S. Andrews, "Cognitive task difficulty analysis using eeg and data mining," in *2017 Conference on Emerging Devices and Smart Systems (ICEDSS)*, IEEE, 2017, pp. 52–57, ISBN: 978-1-5090-5555-5. DOI: 10.1109/ICEDSS.2017.8073658 (cit. on p. 16).

[27] N. Peitek, J. Siegmund, S. Apel, C. Kastner, C. Parnin, A. Bethmann, T. Leich, G. Saake, A. Brechmann, "A look into programmers' heads," *IEEE Transactions on Software Engineering*, vol. 46, no. 4, pp. 442–462, 2020, ISSN: 0098-5589. DOI: 10.1109/TSE.2018.2863303 (cit. on pp. 17, 33, 57, 61).

[28] A. Jbara, D. G. Feitelson, "How programmers read regular code: A controlled experiment using eye tracking," *Empirical Software Engineering*, vol. 22, no. 3, pp. 1440–1477, 2017, ISSN: 1382-3256. DOI: 10.1007/s10664-016-9477-x (cit. on pp. 17, 61).

[29] B. Guerin, A. Matthews, "The effects of semantic complexity on expert and novice computer program recall and comprehension," *The Journal of General Psychology*, vol. 117, no. 4, pp. 379–389, 1990. DOI: 10.1080/00221309.1990.9921144 (cit. on pp. 16, 59).

[30] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, M. Ceccato, "How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes: A series of four experiments," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 96–118, 2010, ISSN: 0098-5589. DOI: 10.1109/tse.2009.69 (cit. on pp. 16, 60).

[31] J. Feigenspan, C. Kastner, J. Liebig, S. Apel, S. Hanenberg, "Measuring programming experience," in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, IEEE, 2012, pp. 73–82, ISBN: 978-1-4673-1216-5. DOI: 10.1109/ICPC.2012.6240511 (cit. on p. 16).

[32] S. Ajami, Y. Woodbridge, D. G. Feitelson, "Syntax, predicates, idioms — what really affects code complexity?" *Empirical Software Engineering*, vol. 24, no. 1, pp. 287–328, 2019, ISSN: 1382-3256. DOI: 10.1007/s10664-018-9628-3 (cit. on pp. 16, 38, 57, 61, 63).

[33] A. Takang, P. Grubb, R. Macredie, "The effects of comments and identifier names on program comprehensibility: An experimental investigation," *J. Prog. Lang.*, vol. 4, pp. 143–167, 1996 (cit. on p. 16).

[34] D. Lawrie, C. Morrell, H. Feild, D. Binkley, "Effective identifier names for comprehension and memory," *Innovations in Systems and Software Engineering*, vol. 3, no. 4, pp. 303–318, 2007, ISSN: 1614-5046. DOI: 10.1007/s11334-007-0031-2 (cit. on pp. 16, 57).

[35] S. Nielebock, D. Krolikowski, J. Krüger, T. Leich, F. Ortmeier, "Commenting source code: Is it worth it for small programming tasks?" *Empirical Software Engineering*, vol. 24, no. 3, pp. 1418–1457, 2019, ISSN: 1382-3256. DOI: 10.1007/s10664-018-9664-z (cit. on pp. 16, 59).

[36] V. Ramalingam, S. Wiedenbeck, "An empirical study of novice program comprehension in the imperative and object-oriented styles," in *Papers presented at the seventh workshop on Empirical studies of programmers - ESP '97*, S. Wiedenbeck, J. Scholtz, Eds., New York, New York, USA: ACM Press, 1997, pp. 124–139, ISBN: 0897919920. DOI: 10.1145/266399.266411 (cit. on p. 16).

[37] G. Salvaneschi, S. Amann, S. Proksch, M. Mezini, "An empirical study on program comprehension with reactive programming," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, S.-C. Cheung, A. Orso, M.-A. Storey, Eds., New York, NY, USA: ACM, 2014, pp. 564–575, ISBN: 9781450330565. DOI: 10.1145/2635868.2635895 (cit. on p. 16).

[38] W. Lucas, R. Bonifácio, E. D. Canedo, D. Marcílio, F. Lima, "Does the introduction of lambda expressions improve the comprehension of java programs?" In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, I. Machado, R. Souza, R. S. Maciel, C. Sant'Anna, Eds., New York, NY, USA: ACM, 2019, pp. 187–196, ISBN: 9781450376518. DOI: 10.1145/3350768.3350791 (cit. on p. 16).

[39] G. K. Rambally, "The influence of color on program readability and comprehensibility," in *Proceedings of the seventeenth SIGCSE technical symposium on Computer science education - SIGCSE '86*, J. C. Little, L. N. Cassel, Eds., New York, New York, USA: ACM Press, 1986, pp. 173–181, ISBN: 0897911784. DOI: 10.1145/5600.5702 (cit. on pp. 16, 59).

[40] D. Asenov, O. Hilliges, P. Müller, "The effect of richer visualizations on code comprehension," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, J. Kaye, A. Druin, C. Lampe, D. Morris, J. P. Hourcade, Eds., New York, NY, USA: ACM, 2016, pp. 5040–5045, ISBN: 9781450333627. DOI: 10.1145/2858036.2858372 (cit. on p. 16).

[41] T. R. Beelders, J.-P. L. Du Plessis, "Syntax highlighting as an influencing factor when reading and comprehending source code," *Journal of Eye Movement Research*, vol. 9, no. 1, 2016. DOI: 10.16910/jemr.9.1.1 (cit. on p. 16).

[42] R. Rosenthal, R. L. Rosnow, *Essentials of behavioral research: Methods and data analysis*, 3. ed., ser. McGraw-Hill higher education. Boston: McGraw-Hill, 2008, ISBN: 0073531960 (cit. on p. 17).

[43] A. Jedlitschka, M. Ciolkowski, D. Pfahl, "Reporting experiments in software engineering," in *Guide to advanced empirical software engineering*, F. Shull, Ed., Springer, 2008, pp. 201–228, ISBN: 978-1-84800-043-8 (cit. on p. 17).

[44] B. A. Kitchenham, *Evidence-Based Software Engineering and Systematic Reviews*, ser. Chapman & Hall / CRC Innovations in Software Engineering and Software Development Series. Boca Raton: CRC Press, 2015, vol. v.4, ISBN: 978-1-4822-2865-6. [Online]. Available: http://gbv.eblib.com/patron/FullRecord.aspx?p=4742741 (cit. on pp. 18, 22, 23).

[45] X. Zhou, Y. Jin, H. Zhang, S. Li, X. Huang, "A map of threats to validity of systematic literature reviews in software engineering," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, 2016, pp. 153–160, ISBN: 978-1-5090-5575-3. DOI: 10.1109/APSEC.2016.031 (cit. on pp. 18, 22).

[46] B. Cartaxo, G. Pinto, S. Soares, "Rapid reviews in software engineering," in *Contemporary Empirical Methods in Software Engineering*, M. Felderer, G. H. Travassos, Eds., Cham: Springer International Publishing, 2020, pp. 357–384, ISBN: 978-3-030-32488-9. DOI: 10.1007/978-3-030-32489-6{\textunderscore}13 (cit. on p. 18).

[47] C. Wohlin, "An evidence profile for software engineering research and practice," in *Perspectives on the future of software engineering*, J. Münch, K. Schmid, Eds., Berlin and Heidelberg: Springer, 2013, pp. 145–157, ISBN: 978-3-642-37395-4. DOI: 10.1007/978-3-642-37395-4{\textunderscore}10 (cit. on pp. 18, 24).

[48] S. Biffl, M. Kalinowski, F. Ekaputra, A. A. Neto, T. Conte, D. Winkler, "Towards a semantic knowledge base on threats to validity and control actions in controlled experiments," in *Proceedings of the 8th ACMIEEE International Symposium on Empirical Software Engineering and Measurement*, M. Morisio, Ed., ser. ACM Digital Library, New York, NY: ACM, 2014, pp. 1–4, ISBN: 9781450327749. DOI: 10.1145/2652524.2652568 (cit. on pp. 18, 45).

[49] Neto, Amadeu Anderlin, and Tayana Conte, "Identifying threats to validity and control actions in the planning stages of controlled experiments," in *SEKE*, 2014, pp. 256–261. [Online]. Available: https://ksiresearch.org/seke/seke14paper/seke14paper_129.pdf (cit. on p. 18).

[50] D. S. Cruzes, T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *2011 International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2011, pp. 275–284, ISBN: 978-1-4577-2203-5. DOI: 10.1109/ESEM.2011.36 (cit. on p. 22).

[51] J. Corbin, A. Strauss, *Basics of Qualitative Research (3rd ed.): Techniques and Procedures for Developing Grounded Theory*. 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc, 2008, ISBN: 9781412906449. DOI: 10.4135/9781452230153 (cit. on p. 22).

[52] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*, ACM Press, 2014 (cit. on p. 23).

[53] B. Adelson, "When novices surpass experts: The difficulty of a task may increase with expertise," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 10, no. 3, pp. 483–495, 1984 (cit. on pp. 33, 58).

[54] C. S. Yu, C. Treude, M. Aniche, "Comprehending test code: An empirical study," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2019, pp. 501–512, ISBN: 978-1-7281-3094-1. DOI: 10.1109/ICSME.2019.00084 (cit. on pp. 33, 58).

[55] S. Wiedenbeck, "Novice/expert differences in programming skills," *International Journal of Man-Machine Studies*, vol. 23, no. 4, pp. 383–390, 1985 (cit. on pp. 33, 59).

[56] P. A. Orlov, R. Bednarik, "The role of extrafoveal vision in source code comprehension," *Perception*, vol. 46, no. 5, pp. 541–565, 2017. DOI: 10.1177/0301006616675629 (cit. on pp. 33, 60).

[57] F. Medeiros, G. Lima, G. Amaral, S. Apel, C. Kästner, M. Ribeiro, R. Gheyi, "An investigation of misunderstanding code patterns in c open-source software projects," *Empirical Software Engineering*, vol. 24, no. 4, pp. 1693–1726, 2019, ISSN: 1382-3256. DOI: 10.1007/s10664-018-9666-x (cit. on pp. 33, 58).

[58] J. Siegmund, C. Kästner, J. Liebig, S. Apel, S. Hanenberg, "Measuring and modeling programming experience," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1299–1334, 2014 (cit. on p. 33).

[59] S. Jessup, S. M. Willis, G. Alarcon, M. Lee, "Using eye-tracking data to compare differences in code comprehension and code perceptions between expert and novice programmers," in *Proceedings of the 54th Hawaii International Conference on System Sciences*, T. Bui, Ed., ser. Proceedings of the Annual Hawaii International Conference on System Sciences, Hawaii International Conference on System Sciences, 2021 (cit. on pp. 33, 57).

[60] D. Hendrix, J. H. Cross, S. Maghsoodloo, "The effectiveness of control structure diagrams in source code comprehension activities," *IEEE Transactions on Software Engineering*, vol. 28, no. 5, pp. 463–477, 2002, ISSN: 0098-5589. DOI: 10.1109/TSE.2002.1000450 (cit. on pp. 33, 58).

[61] T. V. Ribeiro, G. H. Travassos, "Attributes influencing the reading and comprehension of source code – discussing contradictory evidence," *CLEI Electronic Journal*, vol. 21, no. 1, 2018. DOI: 10.19153/cleiej.21.1.5 (cit. on pp. 35, 58, 61).

[62] D. J. Gilmore, T. Green, "Comprehension and recall of miniature programs," *International Journal of Man-Machine Studies*, vol. 21, no. 1, pp. 31–48, 1984, ISSN: 00207373. DOI: 10.1016/S0020-7373(84)80037-1 (cit. on pp. 38, 63).

[63] M. K.-C. Yeh, Y. Yan, Y. Zhuang, L. A. DeLong, "Identifying program confusion using electroencephalogram measurements," *Behaviour & Information Technology*, pp. 1–18, 2021 (cit. on pp. 38, 63).

[64] E. R. Iselin, "Conditional statements, looping constructs, and program comprehension: An experimental study," *International Journal of Man-Machine Studies*, vol. 28, no. 1, pp. 45–66, 1988, ISSN: 00207373. DOI: 10.1016/S0020-7373(88)80052-X (cit. on pp. 38, 60, 63).

[65] D. Binkley, D. Lawrie, S. Maex, C. Morrell, "Identifier length and limited programmer memory," *Science of Computer Programming*, vol. 74, no. 7, pp. 430–445, 2009, ISSN: 01676423. DOI: 10.1016/j.scico.2009.02.006 (cit. on pp. 38, 58, 63).

[66] P. G. Weintraub, "The importance of publishing negative results," *Journal of Insect Science*, vol. 16, no. 1, p. 109, 2016 (cit. on p. 40).

[67] R. F. Paige, J. Cabot, N. A. Ernst, "Foreword to the special section on negative results in software engineering," *Empirical Software Engineering*, vol. 22, no. 5, pp. 2453–2456, 2017 (cit. on p. 40).

[68] A. Borji, "Negative results in computer vision: A perspective," *Image and Vision Computing*, vol. 69, pp. 1–8, 2018 (cit. on p. 40).

[69] Sheppard, Curtis, Milliman, Love, "Modern coding practices and programmer performance," *Computer*, vol. 12, no. 12, pp. 41–49, 1979, ISSN: 0018-9162. DOI: 10.1109/MC.1979.1658575 (cit. on p. 57).

[70] M. Ceccato, M. Di Penta, P. Falcarin, F. Ricca, M. Torchiano, P. Tonella, "A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques," *Empirical Software Engineering*, 2013, ISSN: 1382-3256. DOI: 10.1007/s10664-013-9248-x (cit. on p. 57).

[71] L. Guerrouj, M. Di Penta, Y.-G. Guéhéneuc, G. Antoniol, "An experimental investigation on the effects of context on source code identifiers splitting and expansion," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1706–1753, 2014, ISSN: 1382-3256. DOI: 10.1007/s10664-013-9260-1 (cit. on p. 58).

[72] A. G. Bateson, R. A. Alexander, M. D. Murphy, "Cognitive processing differences between novice and expert computer programmers," *International Journal of Man-Machine Studies*, vol. 26, no. 6, pp. 649–660, 1987, ISSN: 00207373. DOI: 10.1016/S0020-7373(87)80058-5 (cit. on p. 58).

[73] L. Matzen, M. A. Leger, G. Reedy, "Effects of precise and imprecise value-set analysis (vsa) information on manual code analysis," in *Proceedings 2021 Workshop on Binary Analysis Research*, B. Dolan-Gavitt, Ed., Reston, VA: Internet Society, 2021, ISBN: 1-891562-69-X. DOI: 10.14722/bar.2021.23002 (cit. on p. 58).

[74] W. Barfield, "Skilled performance on software as a function of domain expertise and program organization," *Perceptual and motor skills*, vol. 85, no. 3 Pt 2, pp. 1471–1480, 1997, ISSN: 0031-5125. DOI: 10.2466/pms.1997.85.3f.1471 (cit. on p. 58).

[75] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, S. Tamm, "Eye movements in code reading: Relaxing the linear order," in *2015 IEEE 23rd International Conference on Program Comprehension*, IEEE, 2015, pp. 255–265, ISBN: 978-1-4673-8159-8. DOI: 10.1109/ICPC.2015.36 (cit. on p. 58).

[76] A. Stefik, C. Hundhausen, R. Patterson, "An empirical investigation into the design of auditory cues to enhance computer program comprehension," *International Journal of Human-Computer Studies*, vol. 69, no. 12, pp. 820–838, 2011, ISSN: 10715819. DOI: 10.1016/j.ijhcs.2011.07.002 (cit. on p. 58).

[77] W. BARFIELD, "Expert-novice differences for software: Implications for problem-solving and knowledge acquisition," *Behaviour & Information Technology*, vol. 5, no. 1, pp. 15–29, 1986, ISSN: 0144-929X. DOI: 10.1080/01449298608914495 (cit. on p. 58).

[78] J.-M. Burkhardt, F. Détienne, S. Wiedenbeck, "Mental representations constructed by experts and novices in object-oriented program comprehension," in *Human-Computer Interaction INTERACT '97*, S. Howard, J. Hammond, G. Lindgaard, Eds., Boston, MA: Springer US, 1997, pp. 339–346, ISBN: 978-1-4757-5437-7. DOI: 10.1007/978-0-387-35175-9{\textunderscore}55 (cit. on p. 58).

[79] J.-.-M. Burkhardt, F. Détienne, S. Wiedenbeck, "Object-oriented program comprehension: Effect of expertise, task and phase," *Empirical Software Engineering*, vol. 7, no. 2, pp. 115–156, 2002, ISSN: 1382-3256. DOI: 10.1023/A:1015297914742 (cit. on p. 59).

[80] S. Lee, A. Matteson, D. Hooshyar, S. Kim, J. Jung, G. Nam, H. Lim, "Comparing programming language comprehension between novice and expert programmers using eeg analysis," in *2016 IEEE 16th International Conference on Bioinformatics and Bioengineering (BIBE)*, IEEE, 2016, pp. 350–355, ISBN: 978-1-5090-3834-3. DOI: 10.1109/BIBE.2016.30 (cit. on p. 59).

[81] A. Schankin, A. Berger, D. V. Holt, J. C. Hofmeister, T. Riedel, M. Beigl, "Descriptive compound identifier names improve source code comprehension," in *Proceedings of the 26th Conference on Program Comprehension*, F. Khomh, C. K. Roy, J. Siegmund, Eds., New York, NY, USA: ACM, 2018, pp. 31–40, ISBN: 9781450357142. DOI: 10.1145/3196321.3196332 (cit. on p. 59).

[82] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, B. Sharif, "The impact of identifier style on effort and comprehension," *Empirical Software Engineering*, vol. 18, no. 2, pp. 219–276, 2013, ISSN: 1382-3256. DOI: 10.1007/s10664-012-9201-4 (cit. on p. 59).

[83] J. Johnson, S. Lubo, N. Yedla, J. Aponte, B. Sharif, "An empirical study assessing source code readability in comprehension," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2019, pp. 513–523, ISBN: 978-1-7281-3094-1. DOI: 10.1109/ICSME.2019.00085 (cit. on p. 59).

[84] Hansen, Michael, A. Lumsdaine, and R. L. Goldstone, "An experiment on the cognitive complexity of code," *Proceedings of the Thirty-Fifth Annual Conference of the Cognitive Science Society*, 2013. [Online]. Available: https://pcl.sitehost.iu.edu/papers/hansencode2013.pdf (cit. on p. 59).

[85] N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed, J. I. Maletic, "Developer reading behavior while summarizing java methods: Size and context matters," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, pp. 384–395, ISBN: 978-1-7281-0869-8. DOI: 10.1109/ICSE.2019.00052 (cit. on pp. 59, 62).

[86] I. Crk, T. Kluthe, A. Stefik, "Understanding programming expertise," *ACM Transactions on Computer-Human Interaction*, vol. 23, no. 1, pp. 1–29, 2016, ISSN: 1073-0516. DOI: 10.1145/2829945 (cit. on p. 59).

[87] S. Wiedenbeck, "The initial stage of program comprehension," *International Journal of Man-Machine Studies*, vol. 35, no. 4, pp. 517–540, 1991, ISSN: 00207373. DOI: 10.1016/S0020-7373(05)80090-2 (cit. on p. 59).

[88] ——, "Beacons in computer program comprehension," *International Journal of Man-Machine Studies*, vol. 25, no. 6, pp. 697–709, 1986, ISSN: 00207373. DOI: 10.1016/S0020-7373(86)80083-9 (cit. on p. 59).

[89] V. Fix, S. Wiedenbeck, J. Scholtz, "Mental representations of programs by novices and experts," in *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93*, B. Arnold, G. van der Veer, T. White, Eds., New York, New York, USA: ACM Press, 1993, pp. 74–79, ISBN: 0897915755. DOI: 10.1145/169059.169088 (cit. on p. 60).

[90] R. J. Miara, J. A. Musselman, J. A. Navarro, B. Shneiderman, "Program indentation and comprehensibility," *Communications of the ACM*, vol. 26, no. 11, pp. 861–867, 1983, ISSN: 0001-0782. DOI: 10.1145/182.358437 (cit. on p. 60).

[91] G. Bavota, A. Qusef, R. Oliveto, A. de Lucia, D. Binkley, "Are test smells really harmful? an empirical study," *Empirical Software Engineering*, vol. 20, no. 4, pp. 1052–1094, 2015, ISSN: 1382-3256. DOI: 10.1007/s10664-014-9313-0 (cit. on p. 60).

[92] E. Soloway, K. Ehrlich, "Empirical studies of programming knowledge," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, pp. 595–609, 1984, ISSN: 0098-5589. DOI: 10.1109/TSE.1984.5010283 (cit. on p. 60).

[93] B. E. Teasley, "The effects of naming style and expertise on program comprehension," *International Journal of Human-Computer Studies*, vol. 40, no. 5, pp. 757–770, 1994, ISSN: 10715819. DOI: 10.1006/ijhc.1994.1036 (cit. on p. 60).

[94] N. Pennington, "Stimulus structures and mental representations in expert comprehension of computer programs," *Cognitive Psychology*, vol. 19, no. 3, pp. 295–341, 1987, ISSN: 00100285. DOI: 10.1016/0010-0285(87)90007-7 (cit. on p. 61).

[95] E. Daka, J. Campos, G. Fraser, J. Dorn, W. Weimer, "Modeling readability to improve unit tests," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, E. Di Nitto, M. Harman, P. Heymans, Eds., New York, NY, USA: ACM, 2015, pp. 107–118, ISBN: 9781450336758. DOI: 10.1145/2786805.2786838 (cit. on p. 61).

[96] B. Katzmarski, R. Koschke, "Program complexity metrics and programmer opinions," in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, IEEE, 2012, pp. 17–26, ISBN: 978-1-4673-1216-5. DOI: 10.1109/ICPC.2012.6240486 (cit. on p. 61).

[97] J. A. Saddler, C. S. Peterson, P. Peachock, B. Sharif, "Reading behavior and comprehension of c++ source code - a classroom study," in *Augmented Cognition*, ser. Lecture Notes in Computer Science, D. D. Schmorrow, C. M. Fidopiastis, Eds., vol. 11580, Cham: Springer International Publishing, 2019, pp. 597–616, ISBN: 978-3-030-22418-9. DOI: 10.1007/978-3-030-22419-6{\textunderscore}43 (cit. on p. 61).

[98] S. Wiedenbeck, V. Ramalingam, S. Sarasamma, C. Corritore, "A comparison of the comprehension of object-oriented and procedural programs by novice programmers," *Interacting with Computers*, vol. 11, no. 3, pp. 255–282, 1999, ISSN: 09535438. DOI: 10.1016/S0953-5438(98)00029-0 (cit. on p. 61).

All links were last followed on April 17, 2022.

Appendix A

Evidence Lists

The following sections contain the full lists of evidence for each of the three most common threats and the corresponding scores according to the evidence profile. Higher values mean stronger evidence. Negative values contradict the threat, positive values support it.
Please note that the strength of evidence is judged with respect to a study's value in the context of this work and should not be seen as a judgment of its value to the general scientific community as a whole. Reporting scores for each individual study is done for the sake of transparency and should not be used to attack individual authors.

A.1 Programming Experience

Does the programming experience of developers affect their ability to comprehend code?

Ref	Score	Reasoning
[59]	-4	They found no differences in code description performance between novices and experts. But they found some differences in eye fixations.
[27]	-4	They did not find significant correlations between programming experience and brain activation. However, they mention explain that this difference could be due to the homogeneous participant group.
[11]	-4	They found only a very small correlation between programming experience and understandability. They also did not report p-values so it is unclear whether this correlation was significant.
[8]	-4	They found that comprehension behavior was similar across different experience groups. However, they only had a small sample size and only conducted a descriptive statistical group comparison.
[32]	-4	They found no significant effect of experience on program comprehension. However, they themselves pose that they did not design their experiment to specifically investigate experience.
[34]	-4	They found that work experience does not impact the ability to correctly describe code. However, it did impact confidence.
[69]	-4	They found that in all three experiments, programming experience did not correlate with performance. Examining experience was not the main focus and they only used reproduction of a program to measure comprehension.
[70]	-3	They found that experience had a marginal effect on code comprehension. Furthermore, they did not find a significant interaction of experience with the treatment.

Ref	Score	Reasoning
[57]	-3	They only briefly mention that when separating developers by experience, they found that results are very similar. They do not report any details on the statistical approach used to compare these groups.
[61]	-3	They found no differences between novices and experts but did not do statistical test to confirm their assertions.
[60]	-3	They briefly mention that experience did not interact significantly with the treatment but do not report the statistical methods employed. They also only used students.
[71]	-3	They found that experience did not have a significant effect on the performance of participants when splitting and expanding identifiers. However, they pose that splitting or expanding tasks do not require the understanding of source code.
[72]	3	They found that experts performed better on all tests. They only tested the memory in recall tasks and focused more on problem solving skills than code understanding.
[73]	3	They found that more experienced participants has significantly higher accuracy. However, they only measured experience with regard to the specific programming domain (security) and they found that the effect only occurred when no additional aid was present.
[74]	3	They found differences between novices in experts. However, they only had a small sample size and only used recall tasks to measure comprehension.
[18]	3	They found that experts were less influenced by shorter identifier names than average developers. However, they only mention this in passing and do not provide details on any direct statistical tests used to prove this assertion.
[54]	3	While they found that experience with tests improves the comprehension of test code, they did not find any effect w.r.t. reading time. They conclude that the type of experience influences different activities related to comprehension.
[75]	3	They found differences between the reading behavior of novices and experts. However, they did not compare the comprehension performance and only had a small sample size.
[65]	3	Java experience increased correctness significantly. However, they only investigated the recall of identifier names, not complete comprehension.
[76]	3	They found that programming experience influences the comprehension of auditory cues. They focus only on specifically auditory clues and they only used students.
[53]	4	They found that experts and novices form distinct representations of programs and perform differently in comprehension tasks. While it was the main focus of the study, in several cases they found no significant differences.
[77]	4	They found differences in program recall for expert and non-expert programmers. While it was the main focus of the study, the comprehension task only consisted of recalling the correct program order.
[78]	4	They found that when reusing parts of a program to implement a new problem and documenting an existing program, developer expertise was a significant factor. While the study has the correct main focus, the comprehension task is quite unusual.

Ref	Score	Reasoning
[79]	4	They found differences between novices and experts in most areas, but not all.
[10]	4	While it was not the main focus of their study, they found that subjects with more experience make fewer errors when comprehending code.
[15]	4	They found significant differences between novices and experts. However, they only analyzed subjective understandability through ratings.
[80]	4	Clear differences were found between experts and novices when comparing the brain activation during comprehension measured with an EEG. However, their performance w.r.t. correctness and accuracy of their responses showed no significant differences.
[35]	4	In some tasks, experts achieved significantly more correct answers than novices. They conclude that experts and novices behave differently when dealing with comments.
[81]	4	They found that only experienced programmers comprehended source code with longer identifier names more efficiently. Experience was only investigated as a moderator for the treatment effect.
[82]	4	They found a differences in the interaction of experience with the treatment. While the study focused specifically on experts and novices, the type of experiment was exceptionally different to common program comprehension tasks.
[83]	4	While it was not the primary focus of the study, they found that participants with more Java knowledge performed better in comprehension tasks.
[84]	4	They found that both python and general programming experience improved correctness and time of comprehension. However, it was not the main focus of the study and in some cases the observed improvement was not as clear.
[21]	4	They found that novices take longer and are less accurate in finding defects. This was not the main focus and they did not measure the level of comprehension directly.
[85]	4	They did not measure program comprehension performance, but how participants read code. They found that novices and experts looked at different areas.
[86]	4	They found statistically significant interactions between correctness and class level for different EEG measures. However this was just a minor part of the overall study.
[87]	4	They found that experts were better at determining program function. The main focus was on investigating beacons in programs, not investigating the influence of experience.
[39]	4	They found that programming experience had a statistically significant effect on comprehension quiz scores. This was not one of the main hypotheses investigated, however.
[88]	4	They showed significant differences between novices and experts. However, the experiment only employed recall tasks.
[29]	5	The threat was the main focus of the study. They found statistically significant effects of experience.
[55]	5	The threat was the main focus of the study. They found significant differences between novices and experts in reaction times and accuracy.

Ref	Score	Reasoning
[89]	5	The threat was the main focus of the study. The study found that experts had more sophisticated mental representation of abstract characteristics when comprehending code compared to novices.
[30]	5	The threat was the main focus of the study. They found a significant effect of experience on the comprehension level.
[90]	5	Experience was one of the main independent variables analyzed. Overall, experts performed better than novices.
[64]	5	Experience was one of the main five propositions investigated. They found that experts outperformed novices in all three dependent variables.
[56]	5	The main focus was comparing novices with experts in terms of their eye movements. They found that experience significantly interacted with the treatment.
[91]	5	Experience was one of the main variables analyzed. They found statistically significant influences of experience on both time and correctness.
[92]	5	The threat was the main focus of the study. In both experiments, they found that experts performed significantly better than novices.
[93]	5	The threat was the main focus of the study. They observed differences between novices and experts.
[14]	5	The threat was the main focus of the study. In multiple experiments, they showed that there is a difference in the perception of beacons between novices and experts.

A.2 Program Length

Do treatment effects observed in short code snippets also appear in longer snippets or larger code bases?

Ref	Score	Reasoning
[28]	-4	They attempted to confirm that the length of code does not matter and rather that the context is the deciding factor. Some results show that regular (long) and non-regular (short) snippets have different comprehensibility, but for median results they show no difference. Overall, they conclude that the subjects do not achieve lower scores due to high values of LOC.
[94]	-4	Conducted two studies, one with shorter and one with longer snippets to see whether an observed effect still occurs. While they found the effect occurred in both studies, they don't explicitly address the aspect of different lengths again in their second study.
[11]	-4	They calculated the correlations of many different code metrics with understandability. While program length was not the main focus of the study, they found no correlations with any of the metrics they analyzed.
[9]	-3	They found that none of the code metrics correlated significantly with the difficulty of EiPE questions. They only analyzed the number of statements rather than LOC and did not disclose the employed statistical methods.
[32]	-3	Snippets with longer conditionals took more time and had higher error rates than those with shorter ones. However, they find that the overall snippet length was not a significant factor in their experiment. This was only briefly discussed in the threats to validity section and did not include a detailed statistical analysis.
[27]	-3	They found no relationship between lines of code and concentration. However, they mention that the code snippets were designed to be of similar length, so it's unlikely they would be able to identify an effect, even if it was present.
[95]	-3	Number of statements showed no effect but line and identifier length did. Furthermore, they only focused on test code.
[96]	3	While they compared LOC and the opinions of programmers regarding comprehensibility, they did not make a final conclusion as to its validity as a measure. They found that there were correlations with perceptions of complexity, but such a perception might not be useful at all.
[61]	3	While there are some conflicting results whether longer snippets are easier or harder to understand, they always found differences between long and short snippets. They also pose that other characteristics could have influenced the results more than code size.
[97]	4	They compared reading behavior for short and long programs and found that trends occur in longer snippets that do not occur in smaller programs. However this was not the main focus of the study.
[98]	4	They found that the observed effect occurred in smaller programs but not in larger ones. However this was not the main focus of the study.

Ref	Score	Reasoning
[85]	5	The threat was the main focus of the study. They found evidence that experts' reading behavior differs between short and long snippets.
[16]	5	The threat was the main focus of the study. They showed differences in time and accuracy for longer and shorter programs.

A.3 Comprehension Measures

Ref	Score	Reasoning
[62]	-4	The main focus was on comparing declarative vs. procedural programming notations. They found correlations between time and accuracy measures.
[13]	-4	The main focus of the study was to compare cloze with multiple choice questions. They found that for cloze tests where participants were required to understand the snippet, time and accuracy significantly correlated.
[65]	3	They mainly compare the length of identifiers with recall performance. As part of their analysis they compare time and correctness (accuracy) and found no statistically significant effect.
[64]	4	They evaluated the dependent measures before conducting their main analysis. Time and the error measure were not correlated.
[63]	5	They compare the correlations of cognitive load with accuracy, reaction time and self-reported levels of confidence and difficulty (ratings). They only found small correlations and conclude that overall, neither self-reported data nor brainwave activity alone is a reliable indicator of programmers' level of comprehension for all types of code snippets.
[32]	5	They measured the correlation between time and accuracy. They find no strong correlation between the two, indicating that they may reflect different effects. They also found other patterns that support that conjecture.
[24]	5	They found that between self-reported task difficulty (ratings), average normalized oxy measured in fMRI (cognitive load), and fixation duration (visual effort) do not correlate and appear to be measuring different aspects of task difficulty.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature