Technical University of Košice
Faculty of Electrical Engineering and Informatics
Department of Computers and Informatics

Research Methods in Computer Science

Matúš Sulír

September 2026

Research Methods in Computer Science

Author:	Matúš Sulír
Date:	September 2026
Edition:	1st
Publisher:	Technical University of Košice
Address:	Letná 1/9, 042 00 Košice, Slovakia
Document type:	course notes
Form:	online
Copies:	0
URL:	https://sulir.github.io/vmi/1st-ed
Pages:	96
Language:	English

This work was supported by the Slovak Research and Development Agency under the Contract no. APVV-23-0408. It was also supported by project KEGA No. 061TUKE-4/2025 Building Bridges between University and High School ICT Education.

ISBN 978-80-553-5024-0

Table of Contents

Preface

Computer science students are often expected to conduct and report on research, be it in the form of a master's thesis, papers published during their PhD studies, or other research projects. Numerous courses teach the students technical knowledge: to program in various programming languages, design software using diverse paradigms, or understand hardware and networks. Although instructions for their bachelor's theses may contain a few hints about specific technicalities such as citation, none of the courses instruct the students on how to formulate research questions, design an experiment, or write a readable paper targeting their audience.

For this reason, I created the course Research Methods in Computer Science, later renamed to Research Methods in Computer Science and Cybersecurity. It is taught at the master's level in the form of seminars at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice. This book represents course notes and is used as the primary material for students during the whole semester. It covers a wide range of topics: the formulation of research ideas, reviewing literature, asking research questions, executing controlled experiments, performing surveys and benchmarking studies, scientific writing, reviewing, publishing, presentation, and many more. While numerous textbooks about research methods exist, a large portion of them were written for social scientists. A few textbooks focus on specific computer science subfields, but we lack a text explaining research methods used in computer science in general, which this book aims to fix at least partially. My main research area is software engineering, occasionally intersecting with human-computer interaction. Despite trying to be impartial, the text may sometimes be a bit biased toward these two fields.

The course notes are prepared with the concept of a flipped classroom in mind. Instead of spending classes by passive listening to the teacher and then practicing at home without any feedback, students should read an announced part of the learning material at home before the lesson. Then, during the class, they discuss the topic by answering relevant questions, presented in the Exercises section at the end of each chapter. Since some of the exercises require studying additional material or longer reflection, the students should write personal notes outlining their responses during their home preparation, in order to maximize the amount of time spent on discussion during the lesson.

Three larger assignments are also included, focused on systematically finding relevant literature on a selected topic, processing data from a controlled experiment, and reviewing

a paper. They are of a highly practical nature and represent activities that many students will actually encounter, particularly if they decide to continue with the PhD path.

Many topics are only briefly outlined or not mentioned at all in these course notes. Instead of being an exhaustive reference on research methods, I see them as a starting point that helps students understand the underlying principles and then encourages them to explore additional possibilities.

1 Overview

In the first chapter, we will try to distinguish what is (and what is not) research. We will continue with a bird's-eye view of the whole research process.

1.1 Research and Science

Let us start with a few definitions of the word *research*:

> "a systematic investigation to find answers to a problem"
>
> – Burns (2000)

> "creative and systematic work undertaken in order to increase the stock of knowledge"
>
> – OECD (2015)

> "a detailed study of a subject, especially in order to discover (new) information or reach a (new) understanding"
>
> – *Cambridge Dictionary*

Therefore, research is an activity that:

- studies something worth studying (relevance),
- uses a systematic approach (soundness),
- and adds new information to the existing body of knowledge (novelty).

A related term, *science*, is usually defined more narrowly:

> "any system of knowledge that is concerned with the physical world and its phenomena and that entails unbiased observations and systematic experimentation"
>
> – *Encyclopedia Britannica*

> "the systematic study of the structure and behavior of the physical and natural world through observation, experimentation, and the testing of theories against the evidence obtained"
>
> – *Oxford Dictionary*

We can see that science concerns a natural, externally observable world. Parts of this world are systematically observed or manipulated, and, based on the results, new knowledge is produced. Such an approach is called *empirical*, from Greek empeiría (experience), where evidence is formed through senses, using measurement and experimentation.

According to the aforementioned definitions, mathematics is not a science since mathematicians do not observe the external world, and they manipulate only abstract entities they devised. Humanities, such as history, literature, or art, are usually not considered sciences either.

1.2 Computer Science

A question now arises: Is computer science a science? According to Denning (2005), yes – as it meets every criterion. We can devise testable hypotheses, observe the external behavior of computers or the people using them, and construct theories based on these observations.

Abelson (1986) thinks the opposite, as he noted in the opening lecture of the Structure and Interpretation of Computer Programs course:

> "It might be engineering or it might be art, but we'll actually see that computer so-called science actually has a lot in common with magic, and we'll see that in this course. So it's not a science. It's also not really very much about computers. And it's not about computers in the same sense that physics is not really about particle accelerators, and biology is not really about microscopes and Petri dishes."

Note, however, that this view originates in the era when empirical studies, such as experiments with human participants or thorough evaluations using computer benchmarks, were rarer than today. Many research papers of that time described a new idea that the researcher got, along with some implementation details, neglecting evaluation. As one of the voices for a change, a paper advocating evidence-based software engineering as an analogy to evidence-based medicine (B. A. Kitchenham et al., 2004) was published at the beginning of this century.

1.3 Computer Science Subfields

It is difficult to talk about research methods in computer science in general. Computer science is a diverse field, encompassing a large variety of subfields (e.g., embedded systems, cybersecurity, high-performance computing, operating systems, databases, or virtual reality). There have been multiple attempts to divide computer science research into categories, including the Association for Computing Machinery (ACM) Special Interest Groups (SIGs), the ACM Computing Classification System (CCS), and the arXiv category taxonomy (expand the "Computer Science" item). Although the areas certainly overlap to some degree, a majority of researchers specialize in a few narrowly defined topics pertaining to one computer science subfield.

Each subfield has its own typical strategies of advancing knowledge. On one side, there is theoretical computer science. Here, the researcher often starts by defining a theorem, which is split into smaller lemmas. Each lemma is mathematically proved, which finally results in the proof of the main theorem. The research process is thus similar to mathematics as the researchers operate only with abstract notations. However, the proved properties of algorithms, e.g., time complexity, are sometimes empirically verified by executing the program on a computer and measuring the necessary properties.

Human-computer interaction, on the other hand, is people-centric. Traditionally, human participants are observed interacting with a specific hardware or software user interface, and selected quantities are measured.

Some subfields themselves use plenty of different research methods. For example, software engineering papers often describe a new approach designed by the authors, which is subsequently evaluated empirically using simulation, case studies, surveys and interviews, experiments, etc.

Other areas lean toward a commonly used research method. In machine learning, for example, standardized measures such as accuracy or F_1 score are often computed on a popular benchmark for the given problem. This facilitates comparison between multiple approaches but can easily divert attention from problems arising in real-world usage contexts.

1.4 Interdisciplinarity

Just as the border between computer science subfields is blurry, so is often the border of computer science as a whole. Many research projects span multiple branches of science. The knowledge from multiple fields is combined, and new knowledge, related to one or more of these fields, is produced. This is called an *interdisciplinary* approach.

Interdisciplinary research can bring new insights to real-world problems that would be otherwise difficult to solve. On the other hand, the problem of research shallowness often

particularly affects interdisciplinary research. Consider an application of an off-the-shelf machine learning algorithm to weather prediction. Without an expert in meteorology being a part of the research team, there is a risk of misinterpretation of the results and the neglection of methodological or practical issues. Furthermore, while it may be a research contribution to the field of meteorology, we could hardly consider it a contribution to the field of computer science.

1.5 Research Process Overview

Research is an unpredictable activity to a high degree. Despite this, in Figure 1.1, there is an approximation of the journey a computer science researcher can typically take to finish one research project. Be aware that we could add an arrow from almost any activity to any previous activity.

At the beginning, we must have an idea of what the project will aim to accomplish and why this is relevant for other researchers or the general public. Then we start gathering and studying relevant literature. Throughout the rest of the research process, we will get a better understanding of the topic, so we should return to this step regularly.

Although some researchers leave the writing of the paper as one of the last steps, it is beneficial to start writing as soon as we have a general idea and a few relevant research papers. Putting your ideas into writing helps you clarify your thinking, find inconsistencies soon, and prevent forgetting. Writing should ideally occur after each of the next steps, until sending the paper for review.

The following steps are variable and differ between projects. It is possible to create a software or hardware artifact based on a new approach we designed and described in the paper, e.g., an implementation of an algorithm, a new editor plugin, or a novel pointing device. We suppose the designed approach (and thus the artifact that implements it) has some properties (it is more efficient, more usable, etc.), based on which we pose research questions or hypotheses. Alternatively, we can skip the artifact creation and do purely empirical research instead, by asking research questions or posing hypotheses about currently existing artifacts, people, or processes.

After carefully planning the research methods corresponding to the research questions, we collect and process the data and answer these questions based on the data.

It is possible that the purpose of selected research questions was to find shortcomings of existing approaches or our newly-designed artifacts. If this is the case, we can continue by designing a better artifact and repeating a part of the process.

After the paper is finished, we send it for peer review to an appropriate journal or conference, where about three experts will assess its suitability for publication. If the paper is not rejected, we incorporate the reviewers' comments into it, and, depending on the type of the publication, send it again for review and repeat until it is accepted.

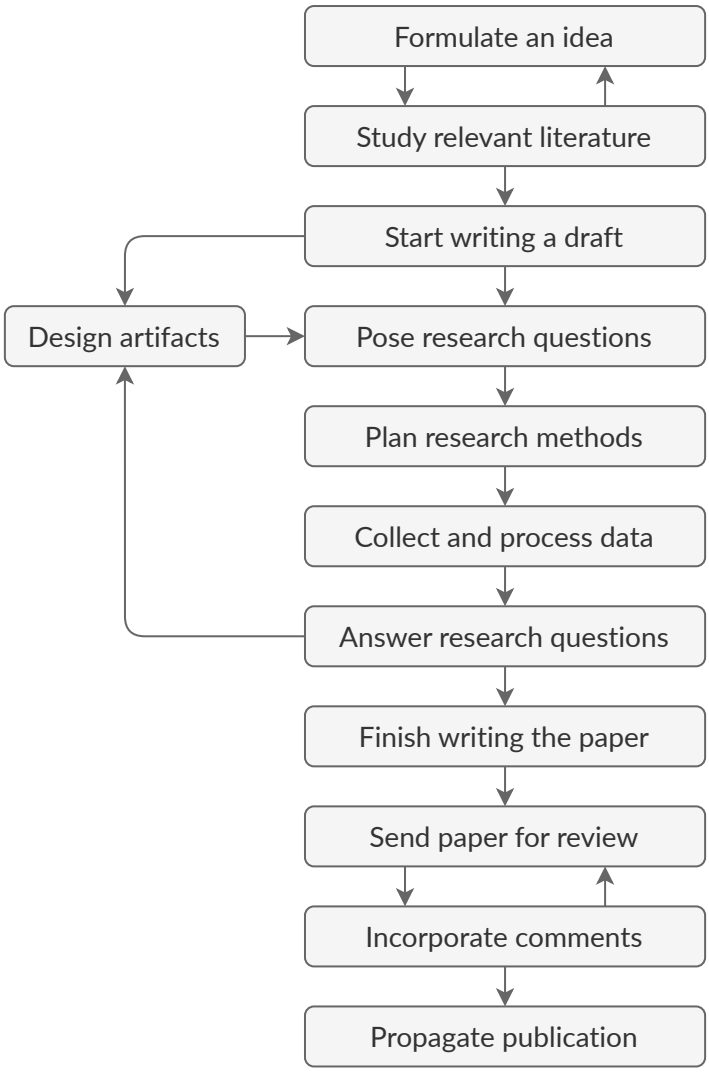

Figure 1.1: An overview of the research process

The paper is then published in the given journal or conference proceedings. This is usually considered the final step, but we should remember to spread the knowledge further, e.g., summarize it in a blog post or archive it on our personal website if it is not freely accessible from the publisher.

In the following chapters, we will discuss the individual steps of the research process, starting with the formulation of ideas and searching for relevant literature.

Exercises

1. Suppose you invented a new algorithm to find the shortest path between two vertices in a graph. Is this research? Is it science? Why or why not?
2. Consider this extreme example: You created a program appending the letter "a" to every 15th occurrence of the letter "z". The letters and the occurrence number are not configurable, and your algorithm uses hacks relying on these constants. Based on your measurements, the program is faster, more memory efficient, and more user-friendly than any other program doing it. Should this be called research?
3. According to you, which of the following documents describe research and why?

    a. Fatal Abstraction (Steimann, 2018)
    b. Business Calendar
    c. Fluorescence SpectraViewer
    d. Bringing Rich Experiences to Memory-Constrained TV Devices
    e. DDD — A Free Graphical Front-End for UNIX Debuggers (Zeller & Lütkehaus, 1996)
    f. On the Dichotomy of Debugging Behavior Among Programmers (Beller et al., 2018)

4. Compare a few papers published at the leading software engineering conference, ICSE, in 1989 (Lubars, 1989; Roman & Cox, 1989; Saeki et al., 1989) and 2024 (Carvalho et al., 2024; Choudhuri et al., 2024; Huang et al., 2024). How do they differ?
5. What computer science subfield(s) do you consider your specialty? For inspiration, you can look at the illustrated Map of Computer Science.
6. Find an example of an interdisciplinary research paper with the experts in two or more domains as co-authors and providing, in your opinion, a valuable contribution to some research field(s). Find another paper where you think the expertise from one field was lacking. One of the domains has to be a part of computer science.
7. Which parts of the research process do you consider the most difficult and why?

2 Forming Ideas

When starting a research project, we need to define what we would like to accomplish: from a general idea to a more specific research goal. This is tightly interconnected with searching and reading works related to our topic. Recall from Figure 1.1 that there is a loop between the idea formulation and studying relevant literature. Neither of these two steps is definitely the first one, as they influence each other.

So, how do researchers come up with a new research idea? The process is usually a chaotic mixture of events, but there are some general guidelines.

2.1 Gaps in Literature

The most obvious way is searching the literature for gaps in knowledge. This seems simple, but the reality is more complicated. First, to determine which piece of knowledge is nonexistent but worth studying, one should have a pretty good understanding of the topic obtained by reading tens or hundreds of papers on it.

Suppose we are already knowledgeable on the topic. How specifically do we derive a new idea from existing papers?

Almost all research articles contain a section called Future Work at the end, or at least a paragraph describing what future ideas the authors envision. Sadly, these ideas may be too general, unrealistically difficult, too incremental, or already done by someone in the meantime. If the paper was published relatively recently, the original authors may as well be working on some of those ideas right now.

A particularly interesting idea is to tabulate certain properties of existing approaches and find missing cells. Consider a hypothetical example about automated source code documentation generators. After reviewing existing documentation approaches, we would find they can be characterized by two *dimensions*: input (with *attributes* "method" and "class") and output (with attributes "sentence" and "paragraph"), as shown in Table 2.1.

Table 2.1: An example of the dimensions of documentation approaches

		Input	
		method	**class**
Output	**sentence**	Foo et al.	Bar et al.
	paragraph	Baz et al.	–

We can see that other researchers have already devised documentation generators that take source code methods or classes as input and produce sentences. Documentation generators producing paragraphs, however, take only methods as input. We could thus try to devise a new documentation approach generating whole paragraphs from classes, provided it was useful and feasible.

Instead of finding a nonexistent combination of attributes, we can even devise completely new attributes or dimensions. In our example, we could explore the possibility of using images as the output of a documentation generator. Other options include generalizing an existing approach, making it more specific, or combining multiple approaches in a nontrivial manner.

If we aim to perform an empirical study instead of designing a new approach, we can apply a ready-to-use framework with a list of dimensions. One of these frameworks, PICOC (B. Kitchenham & Charters, 2007), stands for Population (for example, testers), Intervention (e.g., multi-font syntax highlighting), Comparison (traditional syntax highlighting), Outcome (the number of bugs), and Context (large-scale projects). In this example, we could possibly change the "population" dimension from testers to frontend developers, i.e., determine whether frontend developers produce fewer bugs using multi-font syntax highlighting in a large-scale project.

2.2 Personal Experience

Many ideas come from personal experiences, frustrations, and anecdotal evidence of researchers. For instance, when trying to compile ten different open-source Java projects, five of them might fail. We might start wondering if we were just unlucky or whether this is a general phenomenon. To confirm or disprove this, we may design and execute an empirical study with thousands of such projects.

As researchers often also teach, many papers touch on pedagogical topics. Care should be taken to make such studies really systematic, and to critically admit whether the authors are experts on pedagogy before attempting to publish such a paper.

Many times, researchers in computer science represent potential users of approaches they design or study: they develop software, write documentation in markup languages, use human-computer interaction devices, or use video editing software. As such, they

can think critically about the current state of the given domain. For instance, when filing a bug in an issue tracking system, could the title be generated automatically using language processing methods? Note, however, that research tends to be ahead of practice in many fields. If popular systems do not implement a given feature, it definitely does not mean there is no research in the area.

Researchers often extend their own work. A useful pattern is the iteration between empirical studies and artifact creation. We start with an empirical study that finds problems in the current state of the art. Based on these ideas, we design a new approach. It is then empirically evaluated to confirm if the previously present problems were eliminated. During this evaluation, we usually find new problems. With this knowledge, we design a new, improved version of the approach and continue in a loop.

2.3 Other People

Talking to other people is another source of inspiration for research ideas. For students, one of the most obvious choices is a thesis advisor. In bachelor's and master's studies, the advisors usually already have an idea prepared for the students, ready to be fine-tuned by communication. On a PhD level, the topic provided by an advisor tends to be much more general, and it is expected that the student will actively suggest a more specific idea.

Talking to colleagues, classmates, and professionals from industry about the current problems they face often brings a multitude of great ideas. It is, however, important not to solve only the symptoms of the immediate, obvious issues. Instead, researchers should strive for conceptual improvements, changing the way things work in general. Design thinking is a set of procedures that can help achieve this. One of its components, the "five whys" technique, tries to find the root cause of a problem by asking "why" iteratively. For example, instead of fixing a bug in a code with an automatically generated patch, we might ask: Why was this bug present in the code? After finding it occurred due to a null pointer exception, we ask again: Why did the exception occur? After a few more whys, we might finally find that programmers need a succinct, runtime-safe way to express objects with default behavior in the source code.

Attending academic conferences is another interesting way to get inspiration by seeing a quick overview of what the other researchers in the area are currently working on and informally discussing with peers from various parts of the world.

Exercises

1. Select a published paper in your area of interest. Try to guess how the authors came up with a research idea described in this paper. You can use any sources

and hints: excerpts from the Introduction section, academic search engines, the authors' websites or blogs, or even generative chatbots, given the explanation is logically consistent and all facts are supported by sources. For example, try to find if this is an extension of the author's own previous work or of another paper by different authors.

3 Working with Literature

The techniques for finding relevant literature depend on whether we would like to get an overview of a broad area or find papers related to a specific idea. Learning how to manage a large quantity of papers and read them effectively is also important. This chapter is concluded by an overview of systematic reviews, which are a special kind of studies summarizing literature on a given topic.

3.1 Exploring the Field

For aspiring researchers, such as master's students thinking about their potential thesis topics or first-year PhD students, it is beneficial to see the big picture of their field. This allows them to get an intuition of what was already done years or decades ago, what the current hot topics are, and how the papers in their area typically look.

Research in computer science is typically published as a paper either in a scientific journal or conference proceedings. Therefore, the first step is to prepare a list of the best journals and conferences in the given field. For journals, there exist catalogs filtrable by sub-categories, such as Journal Citation Reports (accessible only from the university network), Scimago Journal Rank, and Research.com journal rankings. For conferences, there is CORE ranking and Research.com conference ranking. The list of top publication venues at Google Scholar mixes journals and conferences together.

For each journal we are interested in, we should find its official homepage, find a few latest issues, scroll through the list of paper titles, and download a few of them to get a feeling of what constitutes top-notch research in the given area.

For conferences, we should also open their official homepages. Most conferences are annual, so we focus on the last two or three years. Top-tier conferences tend to be large and contain multiple tracks, which consist of sessions. It is advantageous to open the main research track's page and write down the names of its sessions (excluding lunches and coffee breaks, of course). Suppose we would like to explore the field of software engineering. At the ICSE 2024 website, we click Program / ICSE Program, expand the "Filter Program" section, and select Tracks / ICSE Research Track. Then we can see the list of days, each containing multiple sessions, in our case: AI & Security 1, Evolution & AI, Testing 1, Analysis 1, Human and Social 1, Generative AI studies, and many others. Constructing such a list helps us understand the current trends in the given field.

For the sessions we are interested in, we can also inspect the list of the corresponding papers.

3.2 Getting an Overview of the Topic

If we narrowed our interest to a more specific topic, such as "time-traveling debugging" or "plant identification using computer vision", we may benefit from reading secondary studies, i.e., articles summarizing existing papers on a given topic. The easiest way to find them is to append the keywords "review", "survey", "systematic literature review", and "systematic mapping study" to the name of our topic and enter them into Google Scholar. Trying synonyms of words in the topic increases the chances of finding the relevant studies. We can also make the topic more specific or general as necessary.

For instance, if we are interested in declarative debugging, we may try:

- "declarative debugging" review
- declarative debugging survey
- algorithmic debugging "systematic mapping study"

Note that some terms are not generally synonyms: *algorithmic* and *declarative* are different terms, but algorithmic and declarative debugging is approximately the same topic. It is thus important to get acquainted with the concepts used by other researchers.

3.3 Searching for Related Papers

Suppose we already have a relatively specific research idea, e.g., "time-traveling debugging using a graphical processor" or "an experimental comparison of cybersecurity education using a traditional and flipped classroom". How do we determine if someone has already published a similar paper and find related papers that we can use for inspiration or for comparison in our Related Work section?

3.3.1 Academic Search Engines

We may use traditional web search engines such as Google or Bing, which return also research papers among other results. However, specialized academic search engines aim to limit the results only to academic literature and offer useful tools such as forward reference lists, which speeds up the process. We will mention some of the more popular ones.

Google Scholar has high document coverage, which is its main benefit and drawback at the same time. Practically all academic documents are indexed – not only journal and

conference papers, but also theses, books, and technical reports. It does not exclude low-quality publications, and occasionally includes also files such as manuals and company whitepapers. No subscription is required.

Semantic Scholar is a tool similar to Google Scholar, with a slightly lower document coverage. The usage is free.

Scopus, on the other hand, is a paid service, so it is accessible only from the subscribed universities' networks or after registration with the university e-mail. Coverage is limited to papers passing certain quality criteria. It offers two features indispensable for systematic literature reviews: advanced search based on tens of properties combinable using a custom query language; and export of all search results into a structured file. Sorting the results by relevance is not on par with the other engines, though.

Web of Science is another paid service. It boasts of strict quality criteria for inclusion. Despite its popularity in many fields, its usefulness for computer science research is severely limited. Waiting times for the indexing of new journals and conference proceedings are as long as a few years. Many conferences are not indexed at all.

OpenAlex is a free alternative website, offering export of the results and an API.

Searching for related papers using AI chatbots usefully supplements academic search engines. However, these tools often do not have full-text access to paywalled articles, so using them as a sole source does not suffice. Care must be also taken to verify if the given results are actually relevant to our prompt.

3.3.2 Searching Process

To find as many related works as possible, it is advised to use a high-coverage academic search engine, such as Google Scholar, with a systematic approach. The general procedure is as follows:

1. Enter a search phrase describing your idea succinctly yet precisely.
2. Read the titles of the documents on the displayed results page.
3. For each document that seems relevant based on title, read its abstract. If the abstract seems relevant, inspect the full text quickly: skim the Introduction, Conclusion, figures, and tables. Search the full text for clues – e.g., if finding an experiment with humans, search the terms such as "human", "subject", "participant", or "enrolled". If the paper is still relevant, download it and save it to your reference manager.
4. Go to the next pages of results until you keep seeing relevant papers, repeating steps 2–3.
5. For each saved paper, look at the backward references. These are the works listed in its References section and thus older than the given article. If any of them seems relevant, perform step 3 on it.

6. For each saved paper, list its forward references, i.e., newer papers that cite it. Google Scholar names this feature "Cited by …". Find and save relevant papers in this list.
7. Repeat steps 5 and 6 on all articles saved in the reference manager but not yet processed. Stop when you seem to be approaching infinite recursion, i.e., the documents you see start repeating too much.
8. Read the full text of the saved papers that seem the most relevant.
9. Repeat steps 1–8 with alternative search terms: synonyms or other aspects of the problem. When forming the search terms, use the knowledge you acquired so far.

Obviously, searching for related works is a long-term process. After performing a five-minute search, we should not claim a paper describing ideas similar to our own does not yet exist.

Tip: We can save time by configuring search keywords for academic search engines in a browser (Firefox, Chrome). For instance, after assigning the keyword "s" to Google Scholar, we can simply type "s motion tracking" into the address bar.

3.4 Accessing Papers behind a Paywall

Some papers are inaccessible for general public without paying. As we will explain later in this book, paying for papers as individuals is not reasonable. Instead, we should apply the following tips.

First, on the publisher's websites, there are always the final versions available, so it is a preferred way if we have access to the given paper. Many universities have pre-paid access to digital libraries such as IEEE Xplore, the ACM Digital Library (free access from 2026), Elsevier ScienceDirect, Springer Link, and Wiley Online Library.

If we are outside the university network, we can try a VPN to remotely access the given resource. An alternative is to register on the given site with a university e-mail, or apply for the CVTI remote access available in Slovakia.

If the publisher's version is inaccessible, we should search for the authors' archived version. For example, on Google Scholar, we can click the "[PDF]" or "All versions" buttons. There are also browser plugins that automate this, most notably Unpaywall.

As a last resort, we can try e-mailing the corresponding author of the paper.

3.5 Reference Management

A reference manager is a program designed to add, edit and view a database of bibliographic citations to research-related documents, along with associated metadata, such as notes, tags, or full text links.

3.5.1 Advantages

Using a reference manager makes a huge difference.

First, we build a database of all papers relevant to us for some reason. Most reference manager have some tagging or categorization capability, so papers related to various projects or topics can be distinguished easily.

Second, papers already read can be marked as such. This may seem useless at first, but after reading at least tens of papers, forgetting is easy.

Third, we can, and absolutely should, make notes about papers. This is related to the previous point – having a paper marked as "already read" is fine, but having a short comment summarizing the main points of the paper is even more useful. Instead of objectively summarizing the facts, we should try to focus on exactly how the paper is relevant to us: How could we use the results? What hinders building upon this paper? Is there any research method used that we could use as inspiration?

Finally, with a reference manager, we can cite effortlessly. Reference managers with BibTeX integration offer the possibility of having one central `.bib` file on a computer, which can be included in each paper via `\bibliography{path/to/file}` in LaTeX. Citing is then as simple as pressing a keyboard shortcut and pasting `\cite{citationKey}` into the source code. Unfortunately, during collaboration this does not work as easily since everyone has a separate BibTeX file.

3.5.2 Software

There are many reference managers available. We will mention a few popular ones.

JabRef is a Java-based desktop application. BibTeX is its native storage format, so all information we edit is actually saved in this structured text file. It offers rich customization options. For example, we can set up automated citation key generation for `bib` file entries, based on a pattern. Adding custom fields with user-defined names to entries is also possible. An older version with a classical look and feel can be downloaded too.

Zotero offers, among other features, PDF file annotation and simple citation import from websites via web browser plugins. Although it uses an opaque data storage format, the database can be synchronized with a `bib` file via the Better BibTeX plugin.

KBibTeX is an alternative suitable mainly for Linux distributions with KDE. For macOS, there is also BibDesk.

As a minimalistic alternative, using a spreadsheet or a structured text file is better than nothing.

3.5.3 Citation Sources

Citation entries for the same article from various websites vary in quality. Although it is possible to download a BibTeX entry from Google Scholar by clicking "Cite" and then "BibTeX" under the given document, such a citation often misses necessary fields, such as the issue number of a journal or a Digital Object Identifier (DOI). Downloading citation information directly from the publisher's website usually provides all available information. Alternatively, we can use sites such as the ACM Guide to Computing Literature, which contain citation records from multiple publishers.

If the downloaded BibTeX record lacks some necessary information, it is sometimes necessary to correct it manually.

3.6 Reading Papers

Now, let us consider an important motivational point: Can someone be a good film director without watching a lot of movies first? How would such a director even know what is a good or bad movie? Similarly, can we be a good researcher without reading a lot of papers first?

However, since research papers are usually long and dense pieces, we should not read them like news articles. Instead, we should adopt the three-pass approach (Keshav, 2007):

The first reading pass should take at most 5-10 minutes:

1. Read the title, abstract, and introduction quickly.
2. Skim headings, tables, figures.
3. Briefly read the conclusion.
4. Mark important references.

If the paper is relevant, we can continue with the second pass. We try to read the rest of the paper from start to end in about one hour (of course, this depends on the length), ignoring not immediately important content and information requiring too much time to comprehend: complicated equations not essential to the topic, mathematical proofs, and low-level details including the description of a tool implementation, hardware specification in benchmarks, etc.

If the paper is really important, e.g., we plan to extend it, we consider applying the described methodology, or we deem it one of the three most relevant papers for our upcoming research projects, the third pass is recommended. During it, we can imagine being the author writing this paper. We try to comprehend all details and resolve ambiguities: How exactly would I perform this (unclear) step in the method? How was

this result computed? What are the shortcomings? How could I do it better? This pass takes multiple hours.

Sometimes, the second and third passes blend together, particularly if we know beforehand that the third pass will be necessary.

3.7 Systematic Literature Studies

A systematic literature study (often called simply a systematic review) is a research study that aims to collect all works relevant to given questions using a rigorous protocol, analyze them, and synthesize results characterizing them. We recognize the two most common types of systematic literature studies:

- *Systematic literature reviews* (SLRs) in their narrow sense analyze the collected articles deeply, often producing quantitative results by aggregating the results of the original (primary) works. They answer specific research questions, such as: What portion of bugs in Android applications can be resolved using automated tools? Which parsing algorithm is the most efficient for context-free languages?
- *Systematic mapping studies* (SMSs) identify sub-topics, categories, trends, or research gaps. They answer broader, less focused research questions, e.g.: How can automated bug resolution techniques be categorized? Which parsing algorithms exist for context-free languages and what are their shortcomings?

Multiple guidelines exist for conducting systematic reviews, including the ones by Carrera-Rivera et al. (2022), B. Kitchenham & Charters (2007), and Wohlin et al. (2024). The best way to get an intuition about how a systematic review is performed is to read specific examples of high-quality papers of this kind, e.g., by Hall et al. (2012), Shahin et al. (2014), Inayat et al. (2015), or Arvanitou et al. (2021) in the area of software engineering or many other papers in the respective fields.

To conduct a systematic review, it is first necessary to pose *research questions* that we would like to answer.

Next, we specify a *search strategy*, which is a process of the retrieval of all works potentially relevant to our research questions. There are three major search methods: manual search, automated search, and snowballing.

Manual search is performed on the lists of all papers published in relevant journals and conference proceedings. Relevant journals and conferences are selected by personal knowledge of the research field or from catalogs. We then manually scan the lists of all published papers in the given year range of these journals and conferences, while selecting only papers relevant for the study based on titles, abstracts, or full texts if necessary.

Automated search is done by entering keywords into the search boxes of digital libraries and/or academic search engines. First, we carefully select keywords, considering all aspects

of our research questions and including synonyms, e.g., for tree and graph-based software visualization it may look similar to `(software OR program) AND visualization AND (tree OR graph)`. A combination of multiple digital libraries or an academic search engine with sufficient coverage is usually used, in a way that maximizes the union of papers covered and minimizes their intersection. It is beneficial if the used websites offer the export of results, otherwise it would be difficult to obtain a deterministic list of papers that can be inspected non-sequentially or by multiple researchers. Similarly to manual search, we select relevant papers based on titles, abstracts, and eventually full texts.

Snowballing means each relevant paper is scanned for backward and forward references. This extends the set of relevant papers. The added papers can again be searched for references, similarly to like a snowball grows.

These search methods are appropriately combined into a search strategy. For instance, we can perform manual search in five journals and seven conferences, then automated search using a database of academic papers, and finally perform backward and forward snowballing with a depth of two on the union of the results from the manual and automated search. During this process, it is necessary to remove duplicates, so that we do not assess one article twice.

To decide which papers are relevant for our study, we use *inclusion and exclusion criteria*. Each paper included in the study has to fulfill all inclusion criteria and must not fulfill any exclusion criterion. A simple example of such a set of criteria is:

- I1: It is a journal or conference paper published between 2015 and 2024.
- I2: The paper includes an empirical evaluation of a tree or graph-based software visualization.
- E1: Editorials, essays, secondary studies, and tool papers are excluded.
- E2: UML-based visualizations are excluded.

The criteria have to be as precise and unambiguous as possible. Ideally, two researchers should be able to independently include/exclude a set of papers with a perfect overlap.

After constructing a list of papers relevant to our study, *data extraction* is performed. We construct a table where we assign each paper numerical, categorical, or other properties (e.g., year: 2020, type: “graph”, evaluation: “controlled experiment”). The set of potential categories for each categorical property is either defined beforehand or refined gradually during the process of data extraction.

Finally, we perform *data synthesis*, where we statistically derive conclusions from the extracted data, such as percentages of papers pertaining to each category. The details of data extraction and synthesis depend on the research questions that we posed in the beginning.

Exercises

1. Choose one of the leading conferences in your favorite subfield of computer science. Open the websites of its last two or three years and find the main research tracks. Compile a list of 10–12 session names per year. If the conference does not have named sessions, list selected paper titles instead.
2. Select a specific enough idea for which you would like to find related papers. For instance, it can be related to your thesis or any paper that you liked. Describe the idea in one or two sentences. Perform the searching process from Section 3.3.2, ignoring step 4, severely limiting or skipping the recursion in step 7, and ignoring step 8. Stop after saving about 10–12 actually relevant papers. Which part of the process was the most efficient in finding relevant papers? Are you satisfied with the results? Do you have any suggestions to improve this process?
3. Use an AI chatbot to find the works related to the idea from the previous exercise. Are the results more or less relevant?
4. Which of the mentioned advantages of reference managers is the most important to you? Can you find any other advantages?
5. Which reference manager do you use and why?
6. Find three papers relevant for you that you have never read at all. Read them with the first pass only. After reading each paper, close it and write down all the information that you remembered. How would you characterize the remembered information? For instance, is it somehow important for your research?
7. (This exercise does not require home preparation.) During your lesson, the teacher will show you a screenshot. Imagine you have an idea that could be explained by a screenshot you see. Your task is to find if a similar idea was already described in research paper(s) and find such a paper. Warning: Do not search for exact texts from the screenshot. Imagine it represents only an abstract idea in your head.

4 Types of Research

When designing a research study, it is necessary to select appropriate research methods based on multiple criteria. In this chapter, we classify types of research according to various characteristics, which will be useful for subsequent discussion throughout this book.

4.1 Quantitative, Qualitative, and Mixed

According to Creswell & Creswell (2018), we distinguish three main research approaches: qualitative, quantitative, and mixed methods research.

Quantitative research typically uses pre-determined procedures and precise numeric measurements. It often includes hypotheses describing relationships between variables, which are objectively tested with the help of statistics. The resulting report tends to have a fixed structure, with standard section names such as "Hypotheses", "Variables", and "Procedure". Typical quantitative research methods include controlled experiments and surveys with closed-ended questions.

In *qualitative* research, the procedure is sometimes not entirely fixed and the collected data are free-form, such as text, images, or videos. Researchers can make subjective interpretations of the collected data and the report does not have a standardized structure. Specific examples of qualitative methods are case studies and unstructured interviews.

Mixed methods research integrates both the quantitative and qualitative approaches. This can be done in multiple ways, but the most common are the convergent, explanatory sequential, and exploratory sequential designs.

In the *convergent mixed methods design*, both quantitative and qualitative data are collected relatively independently, possibly at the same time. Conclusions are drawn by combining the knowledge from both approaches.

Explanatory sequential mixed methods design starts with a quantitative phase, where numerical results are produced or a hypothesis is confirmed or rejected. Subsequently, the researcher tries to explain why such results occurred by conducting qualitative studies. For instance, after finding that a new unit testing technique improves the developers' productivity by 30% in a controlled experiment, we can interview the developers to determine the specific workflows and features that the developers considered useful during their work.

On the other hand, *exploratory sequential mixed methods design* starts with a qualitative phase. Here we try to explore the given topic without a strict, fixed procedure. Based on the collected data, we can find the hypotheses or research questions worth studying, design the instruments (e.g., questions in a questionnaire), or develop an intervention (e.g., a new interaction method with a user interface). In the subsequent quantitative phase, a study is performed based on this newly obtained knowledge.

4.2 Inductive and Deductive

Based on the direction of the relationship between specificity and generality, research can be either inductive or deductive.

Inductive research uses a bottom-up approach as it starts with small, specific observations. By combining and generalizing these pieces of information, a more general theory is developed gradually. As an example, we can mention the grounded theory of self-organizing agile teams (Hoda et al., 2013 and related papers).

Deductive research is top-down: it starts with a general, high-level theory. We formulate more specific hypotheses that should be true if the whole theory is true and test these hypotheses.

4.3 Philosophical Worldviews

Although this is rarely mentioned explicitly in papers, the authors have their philosophical stance toward how our world functions and how research can uncover its laws. Five common philosophical worldviews are positivism, postpositivism, constructivism, transformativism, and pragmatism.

According to *positivists*, there exists an absolute, objective truth. They aim to establish cause-and-effect relationships between variables. The purpose of research is to find such relationships solely through empirical studies and the confirmation of hypotheses.

Postpositivism is similar to positivism, but it accounts for imperfections and bias in the observations and measurements. Therefore, according to postpositivists, hypotheses and theories are never actually confirmed with absolute certainty. The more we test them without rejection, the more confident we are about their truthfulness. Both positivists and postpositivists prefer quantitative research methods and deductive thinking.

According to *constructivists*, the reality is subjective. Therefore, researchers should take the perspectives of different participants into account. Qualitative research methods should be applied, so we can better understand the richness of various views, and construct new theories based on the collected data. Constructivists thus prefer inductive thinking.

The *transformative* worldview comes from the social sciences, where the aim of the researchers is to focus on minorities that are often overlooked during classical quantitative research and suggest transformations so that their situation improves. The transformative worldview is relatively rare in computer science research, compared to the other stances.

Pragmatists combine qualitative, quantitative, and mixed methods research approaches as necessary. They do not see the world as a set of universally applicable laws. Instead, they try to find practical solutions fitting particular specific contexts.

4.4 Basic and Applied

We can also categorize research as basic or applied. Note that this categorization is rather artificial, and the distinction is often not clear.

Basic research (also called pure or fundamental) aims to obtain new knowledge about the given phenomenon, without any particular practical application in mind. It is mainly driven by the researchers' curiosity. This does not mean it is useless – many pieces of knowledge obtained by basic research were successfully applied in practice.

Applied research tries to solve a particular practical problem. The results are immediately actionable and useful to the general public or a selected population.

There exists another related term, *experimental development*, which utilizes the results of basic and applied research to create a novel product with the goal of subsequent commercialization. Together with research, they form the concept of research and development, known as R&D.

4.5 Types of Contribution

According to Wobbrock & Kientz (2016), there are seven possible types of contribution of research papers: theoretical, empirical, methodological, dataset, artifact, survey, and opinion. Although their taxonomy is tailored for the field of human-computer interaction (HCI), it is applicable for many other computer science subfields.

Theoretical contribution represents the creation of a theory, i.e., a proposition explaining certain general phenomena. It is often based on the generalization of a large quantity of evidence in the area.

Empirical contribution consists of a specific empirical study that observes the real world. Each empirical study can partially contribute to the development of a theory.

Methodological contributions improve ways we measure, analyze, and design things. Examples include a new type of questionnaire for measuring user experience or a research method to evaluate human interface devices for visually impaired people.

A *dataset* benefits the research community by offering pre-processed data ready to be analyzed by other researchers and standardized benchmarks to compare multiple approaches.

Artifacts are newly designed and innovative prototypes of software systems, hardware devices, or techniques.

Surveys analyze, categorize, and summarize a large number of existing research works on a specific topic. This should not be confused with questionnaire surveys that pertain to empirical contributions.

Finally, *opinions*, sometimes called essays, aim to lay out the author's thoughts and convince the reader using strong arguments based on scientific evidence and logical justifications.

4.6 Actors, Behavior, and Context

Stol & Fitzgerald (2018) devised a software engineering research methods classification scheme, which is, however, applicable to many other computer science subfields beyond software engineering. It is based on two key dimensions: obtrusiveness, i.e., how much the researchers modify the conditions during the measurement, and generalizability – how much the findings are applicable in other settings.

Research methods are then characterized based on whether they are maximally generalizable over *actors* (A), maximally precisely measure the actors' *behavior* (B), or preserve the maximally realistic *context* (C), thus the name "the ABC of software engineering research". By actors, the authors mean people such as developers or managers, software systems, artifacts (e.g., issues), and techniques. Behavior means system behavior (e.g., performance, storage space) and the persons' productivity, motivation, and other characteristics. Context includes industrial vs. academic settings, specific software projects, or development teams.

4.7 Primary and Secondary

Our final categorization is based on the sources of information used. *Primary* research includes the collection of new data. *Secondary* research analyzes only existing literature or existing data sources. For instance, systematic literature reviews are considered secondary studies. *Tertiary* studies review multiple secondary studies.

Exercises

1. Suppose you would like to determine which of the three database insertion techniques is the most energy efficient. Which research approach would you use – quantitative, qualitative, or mixed methods?
2. Find a qualitative research study that you consider interesting.
3. Find a mixed methods research study in your area of interest. Can convergent, explanatory sequential, or exploratory sequential mixed methods design be clearly recognized in the paper?
4. Which of the mentioned philosophical worldviews do you hold and why?
5. Which philosophical worldview do Shreeve et al. (2023) hold in their paper?
6. Find any paper in your research area whose main contribution type is:

    a. theoretical,
    b. empirical,
    c. dataset,
    d. artifact.

7. Inspect Figure 1 in the ABC framework paper (Stol & Fitzgerald, 2018). Select one of the eight research strategies shown in the circle (experimental simulations, field experiments, …). Find a paper in your area using the selected research strategy.
8. Find any secondary study in computer science that synthesizes quantitative results from a large number (at least 20) of existing papers.

5 Choosing Research Questions and Methods

After we select a specific research topic and idea, it is necessary to specify the research question(s) and/or hypotheses and choose a research method to answer them.

5.1 Research Questions

A *research question* (RQ) is a clearly formulated question that we would like to answer in our study using research methods. In general, a good research question should be:

- Focused: If we have multiple research questions per paper, they should all relate to the main goal.
- Researchable and feasible: We must be able to answer it scientifically and using reasonable resources.
- Specific: A research question should not be vague.
- Complex enough and relevant: We should not study questions that will be useless for sure.

To provide more specific examples, Begel & Zimmermann (2014) and Huijgens et al. (2020) provide lists of questions that industrial software engineers find relevant. Similar lists may exist for other subfields.

Sometimes, a research question (or a hypothesis) is only implicit in a paper. This is the case mainly for papers whose main contribution is an artifact – the research question then might be "Is it feasible to design X?" or "How to design an approach X so that it has some property Y?". However, particularly for empirical studies, we should always state a research question explicitly.

5.2 Hypotheses

A *hypothesis* is a declarative sentence whose truthfulness is yet unknown. It often specifies a relationship between variables, e.g., "A high-contrast mouse cursor improves the clicking precision on a target rectangle on a screen." A hypothesis must be falsifiable, which means we can reject it using an empirical research method.

Any hypothesis can be written as a research question, but the reverse does not always hold. For instance, the question "How do developers unit test database modules in procedural languages?" cannot be written as a hypothesis. If a statement is statistically testable, the hypothesis is a preferred form.

A hypothesis is a statement about a more specific phenomenon, such as "When debugging, having dynamic information from Senseo (Rothlisberger et al., 2012) available reduces the time for solving maintenance tasks." In contrast, a *theory* is a proposition explaining a certain general phenomenon. It is usually based on the generalization of a large quantity of evidence in the area. For example, an information foraging theory for debugging (Lawrance et al., 2013), explains how the developers navigate the program, similarly to a predator following prey, using concepts such as scent, proximal cues, or topology.

5.3 Operationalization

We know that a research question or a hypothesis should be specific and precisely defined. However, even well-stated hypotheses and RQs contain terms with many possible interpretations. Consider the hypothesis:

> Using a time-traveling debugger improves developers' efficiency in fixing bugs.

There exist multiple time-traveling debuggers with different features. Many people consider themselves developers: from high-school students coding for fun to senior developers in corporations. What actually constitutes efficiency is a notoriously debated problem.

We thus have to provide *operational definitions* of important terms. An operational definition expresses how the given term will be interpreted in this specific study, i.e., how it will be measured, observed, or applied.

For example, we will use a specific time-traveling debugger TimeDebugX, implementing certain features. By "developers", we will understand professional programmers with at least one-year industrial Java experience. We will measure "efficiency" as the number of successfully solved bug-fixing tasks per hour.

5.4 Choosing a Research Method

Many beginning researchers make a mistake of selecting a research method without having any specific hypothesis or RQ in mind. For instance, they design a new approach, which they would like to empirically evaluate. They start thinking about a questionnaire survey as a relatively easy way to collect data from multiple respondents. They start adding many questions to the survey, ranging from demographic details of every kind to

many unclear and suggestive questions about how the respondents liked various aspects of the newly designed system.

This is definitely a wrong practice. When doing research, we should always *define RQs/hypotheses first* and only then proceed to choosing research methods. Otherwise, we risk that the method will answer only questions that are irrelevant or will not reliably answer any research question at all.

There are multiple guides and overviews suggesting the most suitable research methods for given types of research questions, e.g., by Wohlin & Aurum (2015) or Easterbrook et al. (2008). In the next chapters, we will describe some of the frequently used research methods, including controlled experiments, surveys, interviews, case studies, and benchmarking studies among others. For each method, we will mention examples of RQs it is suitable for answering. However, in practice, the methods and their principles are sometimes not clearly separated, so we may use, e.g., the concept of sampling known from surveys in a benchmarking study.

During a research project, it is often beneficial to use multiple methods or data sources in a practice called *triangulation*. For example, we can obtain requirements for our new approach using a survey, assess its machine-based efficiency by a benchmark, find its advantages and disadvantages in a case study, and finally assess its human-based efficiency using a controlled experiment with humans. This increases the validity of the study and provides a more comprehensive understanding of the subject.

Exercises

1. Which of the following are good research questions and why?
    a. Why do students fail in the operating systems bachelor's courses?
    b. How to determine if a program written in C terminates without running it?
    c. What proportion of C++ Gists on GitHub is compilable without downloading third-party libraries?
2. Which of the following are good hypotheses? Why?
    a. Java is a fast programming language.
    b. An LL parser is faster than an LR parser for converting JSON into an in-memory tree structure.
3. Find a paper in your field that explicitly lists operational definitions of terms used in a hypothesis or a research question.

6 Controlled Experiments

In this chapter, we explain one of the fundamental research methods, controlled experiments. A controlled experiment is typically used to answer research questions in the form "Does X cause Y?" or "Is A more efficient/precise than B?".

6.1 Correlation vs. Causation

Suppose we would like to know what the relationship is between playing platform games and kids' typing speed on a computer keyboard. We start asking a very large random sample of kids whether they play platform games. During the study, we also measure the typing speed of each participant using a standardized test. As a result, data similar to Table 6.1 is produced, with hundreds of rows.

Table 6.1: A sample of the collected data on the relation between playing platform games and typing speed

Participant ID	**Plays platform games**	**Typing speed (words/min)**
1	false	32
2	false	21
3	false	25
4	true	65
5	true	53
6	true	67
...	...	...

For all participants not playing platform games, the mean is 27 words per minute, while for the playing ones, a mean of 62 was computed. Platform game players thus seem to be markedly faster at typing. Suppose we report our findings in a research paper that gets published. After some time, a famous tabloid comes with a flashy headline:

> "Playing platform games improves typing speed", say researchers.

Based on this fantastic news, many parents sign their kids up for a platform gaming club. Every week, they play Super Mario, Crash Bandicoot, Seiklus, and other popular

platformers. Sadly, after a year of playing, the parents find out there is absolutely no difference in their kids' typing speed.

Why is this the case? Clearly, there is a difference between correlation and causation. The study showed there is a *correlation*: "Kids who play platform games tend to type faster". However, it did not show *causation* at all: "Playing platform games causes kids to type faster."

The question arises about what is the true causation in our example. Is it reversed, i.e., typing faster causes kids to play computer games? In general, reverse causality is possible, but in our specific example, this does not seem plausible. Instead, we should look at other hidden variables that were not measured in our study. During interviews with the participants, we might find out that practically all kids that play platform games also chat using various instant messaging applications. In Table 6.2, there is a new column "chats", representing a *confounding factor*, which is a variable related both to the incorrectly presumed cause (playing platform games) and the outcome (typing speed).

Table 6.2: Internet chatting as a confounding factor

Participant ID	**Plays platform games**	**Chats**	**Typing speed (words/min)**
1	false	**false**	32
2	false	**false**	21
3	false	**false**	25
4	true	**true**	65
5	true	**true**	53
6	true	**true**	67
...	...	...	...

Upon closer investigation, we might find that the reality is even more complicated, as the root cause of both game-playing and chatting is that a given kid has a dedicated computer at home. This situation is displayed in Figure 6.1.

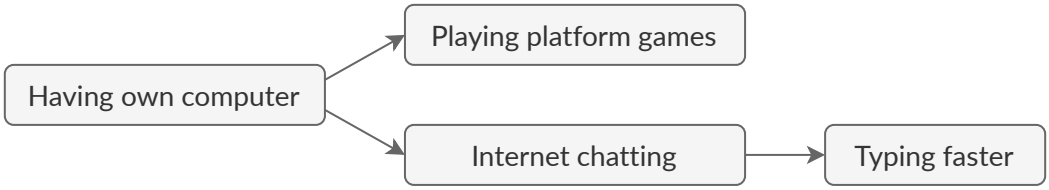

Figure 6.1: Probable true causation; arrows represent causality

In general, there may be many confounding factors. We could carefully search for such confounding factors and take them into account during statistical calculations. This

process is called *controlling for* variables. However, if practically possible, a much more valid approach should be applied instead: a controlled experiment.

A *controlled experiment* is based on the idea that we keep all conditions fixed, while manipulating only the presumed cause. In a very common type of a controlled experiment, a *randomized controlled trial*, we randomly divide a set of subjects (e.g., people, programs) into two or more groups. Each group receives a different treatment; if a group receives no treatment at all (or a placebo), it is called a *control group*. At a specified time(s), we measure the outcome for each subject and compare the results between the groups. This approach works, particularly with a large enough number of subjects, because randomization diffuses the subjects having various characteristics into all groups.

In our example, we could divide the kids (preferably the ones not playing platform games regularly) randomly into two groups. The first one will play platform games for two hours a week, while the second one will not play platform games at all. After a few months, we compare the typing speed of the groups.

6.2 Variables

Before performing a controlled experiment, we need to define variables. A *variable* in research is a measurable, observable, or manipulable characteristic that can vary. An *independent variable* is a condition which we manipulate in the controlled experiment. A *dependent* variable is an outcome which we measure or observe. Its name is derived from the fact that we hypothesize it depends on the value of the independent variable. In some cases, a *mediating* variable, influenced by the independent variable and affecting the dependent variable, can stand between these two variables and explain the relationship.

Each variable has a *scale*. There are four possible scales:

- *Nominal*: one of n possible values called *levels*, without any particular order, such as Linux/Windows. A nominal variable with two levels is called *dichotomous* or *binary*.
- *Ordinal*: one of n values that can be ordered but not subtracted, e.g., primary, secondary, or tertiary education.
- *Interval*: it has equal differences between subsequent values, e.g., a calendar date.
- *Ratio:* a ratio of two values is computable, with a meaningful zero (e.g., time spent on development in hours). It can be further characterized by its statistical distribution, usually normal/non-normal.

The list is ordered from the most general scale to the most specific one, so if an appropriate statistical test exists only for a more general scale, it can be applied instead.

In our example experiment, an independent variable of a nominal scale is playing/not playing platform games. A dependent variable of a ratio scale is the typing speed in words per minute.

6.3 Hypotheses

Next, we define a null and alternative hypothesis. A *null hypothesis*, denoted H_0, assumes there will be no difference between the groups in the experiment, which means changing the value of the independent variable does not cause a change of the dependent variable. An *alternative hypothesis* (H_1 or H_a) stipulates a difference between the groups will be present, so there is a causal relationship between the independent and dependent variable.

In our example, the null and alternative hypotheses could be defined as follows:

- H_0: Playing platform games makes no difference in the typing speed of kids.
- H_1: Playing platform games makes a difference in the typing speed of kids.

The alternative hypothesis in our example is a *two-tailed hypothesis* since it considers a change in both directions; the difference could be either positive or negative. A *one-tailed hypothesis* would consider only the specified direction and deny the possibility of the opposite one, e.g., "Playing platform games improves the typing speed of kids." We should generally use a two-tailed hypothesis unless we have good reasons not to do so, such as when the opposite change is impossible or irrelevant.

6.4 Experimental Design

An *experimental design* formally defines how the experiment will be executed. It tells us how the assignment of the subjects to groups will be performed. It also specifies how many times and in what order the treatment will be administered and the outcome measured.

Based on the randomness of the assignment of the treatments to the subjects, we distinguish:

- *quasi-experiments*, where assignment is performed using some convenient criterion, e.g., the groups will consist of employees working in given teams or students attending given classes
- and *true experiments*, where the assignment is random.

From the perspective of the sequence of treatment administration and measurement events, there exist multiple experimental designs. The most common ones are:

- the *pretest–posttest design*, in which the value of the dependent variable is measured both before and after the administration of the treatment
- and the *posttest-only design*, where we measure the outcome only after the treatment.

Table 6.3: Assignment of treatments to subjects in a between-subject design

Table 6.4: Group 1

Subject ID	Treatment
1	1
3	1
5	1
...	...

Table 6.5: Group 2

Subject ID	Treatment
2	2
4	2
6	2
...	...

Another categorization specifies the number of treatments each subject receives. In a *between-subject design*, each subject is assigned to one specific group during the whole experiment, receiving only one treatment. Table 6.3 shows an example of such a design.

When using a *within-subject design*, each subject receives a treatment, and then the outcome is measured. Then the same subject receives another treatment, and the outcome is measured again. If there are more treatments, these steps are repeated. Table 6.6 displays an example of the assignment of the subjects to treatments in a within-subject design with two treatments. Note that we used *counterbalancing*, i.e., subject #1 was administered treatment 1 first and then treatment 2, while subject 2 received them in the reversed order. This is done to prevent systematic bias, where one of the treatments would be given an advantage.

Table 6.6: Assignment of treatments to subjects in a within-subject design

Subject ID	First treatment	Second treatment
1	1	2
2	2	1
3	1	2
4	2	1
...	...	...

A between-subject design is usable in almost all situations, but it requires more subjects than the within-subject design. On the other hand, a within-subject design requires fewer subjects, but the learning effect and fatigue are a problem when the subjects are human.

An experiment can have multiple independent variables. The most straightforward way is to have a group for every possible combination of independent variables. For instance, with two dichotomous variables, we would have four groups. This is called a *factorial design*.

6.5 Effect Size

Suppose we execute our example experiment and collect the data. Let us say the mean of the not-playing group in the controlled experiment is 45.1 words/minute, and the mean of the playing group is 45.8. Therefore, we can say that on average, the playing group was 0.7 words/minute – or 1.6% – faster. These are one of the simplest ways to express the *effect size*. Although they are valid and easily comprehensible, there exist standardized metrics of effect sizes, such as Cohen's d, which take variability (standard deviation) into account and simplify the meta-analysis of multiple papers. Kampenes et al. (2007) provide a systematic review of effect sizes commonly used in software engineering, including an overview of the possibilities.

6.6 Statistical Testing

One question remains: Are these two mean values (45.1 and 45.8) sufficient to reject the null hypothesis (no difference) and accept the alternative hypothesis? The answer is no because the difference in means could be observed purely by chance. We need to analyze the whole dataset to find whether the result is statistically significant.

For this, we compute a *p-value*, which is the probability of observing the data at least as extreme as we observed if H_0 was in fact true. The largest acceptable p-value is called a *significance level*, denoted α, and must be stated before the execution of the experiment. The most commonly used value of alpha is 0.05 (i.e., 5%). If $p < \alpha$, we reject the null hypothesis, otherwise we fail to reject it.

The p-value is computed by an appropriate statistical test. To select a test, we can use, e.g., a table by UCLA or by Philipp Probst. Each test should be applied only in situations when its assumptions are met. For instance, the independent-samples t-test has multiple assumptions: the data should be approximately normally distributed, there should be no significant outliers, etc. Many statistical tests are implemented in the libraries of programming languages, such as R or Python.

Let us say that in our example, the computed p-value would be 0.67. This means we could not reject the null hypothesis, so we would not show there is any effect of playing platform games on typing speed. If the p-value was, for instance, 0.04, we would reject the null hypothesis and show an effect.

In Table 6.7, we plotted the computed result of the experiment (rejection of H_0 or a failure to do so) against the reality, which is unknown. Correctly rejecting or not rejecting H_0 is called a true positive and a true negative, respectively. Accepting the alternative hypothesis, i.e., showing an effect, while there is in fact no effect, is called a Type I error. The maximum acceptable probability of a Type I error is α, as we already mentioned.

Table 6.7: Possible outcomes of an experiment compared to the reality

		In reality, H_0 is	
		true (no effect)	**false (effect)**
We reject H_0	**no**	true negative	Type II error
	yes	Type I error	true positive

There is also another type of error in Table 6.7: a Type II error. It means we failed to show an effect when there actually is one. The probability of a Type II error is denoted β (beta), with a common value of 0.2. To mitigate a Type II error, we should perform the experiment with a large enough number of subjects and choose a suitable statistical test, so the experiment will have enough *power* $(1 - \beta)$. It is possible to estimate the number of subjects required for a given statistical test to have the given power using a process called power analysis.

6.7 Threats to Validity

Every research project has issues that could negatively affect its validity, which we should mention in the report in a section called "Threats to Validity". In controlled experiments, these are some common types of validity which can be threatened:

- *Construct validity*: Did we choose the right variables and measures?
- *Internal validity*: Are the results affected by other factors beyond causality?
- *External validity*: Are the results generalizable to other situations?

In addition to listing the validity threats, we should also mention how we tried to mitigate them or why it was not possible.

6.8 Complete Example

To show the whole picture, now we will describe a complete example of an imaginary controlled experiment, along with the data analysis using the R statistical programming language. Suppose we have the following *research question (RQ)*: Does the syntax highlighting type (none, mild, strong) affect the developers' speed when locating a code fragment?

6.8.1 Hypotheses

We formulate the null and alternative hypothesis:

- H_0: There is no difference in the developers' mean code location speed when using none, mild, or strong syntax highlighting.
- H_1: There is a difference in the developers' mean code location speed when using none, mild, or strong syntax highlighting.

We will test with $\alpha = 0.05$.

6.8.2 Variables

There is one *independent* variable: syntax highlighting type. It is nominal (categorical) with 3 levels. The *dependent* variable is of the ratio type (code location time in seconds).

6.8.3 Design

One developer cannot search for the same code fragment multiple times using different highlighting styles because of a learning effect. We could use different code locations and source files for each syntax highlighting type, but different locations/files can be of varying difficulties. We also had a large pool of participants available. Therefore, we used the between-subject posttest-only design.

6.8.4 Procedure

We recruited 33 final-year computer science Master's students. We divided them randomly into 3 groups (none, mild, strong).

They were given 10 source code files. For each source code file, there were two tasks specified, such as: "Find the first ternary operator in the function sum()" or "Find the exception handling code in a function which receives a JSON object".

After reading each task, the subject pressed a shortcut to display the code file. At this moment, the timer started. After finding the fragment, the participants placed a cursor on it and pressed another shortcut – if the location was correct, the timer stopped. The total time was recorded for each subject.

6.8.5 Results

The measured values are located in a CSV file, of which we show only the first few rows for brevity.

```
results <- read.csv("data/experiment.csv", header = TRUE)
head(results)
```

```
   group time
1   none 1241
2   none 1215
3   mild 1060
4   mild 1084
5 strong 1005
6 strong 1005
```

An overview of the differences between groups in the form of a box plot is in Figure 6.2.

```
results$group <- factor(results$group, c("none", "mild", "strong"))
boxplot(time ~ group, results)
```

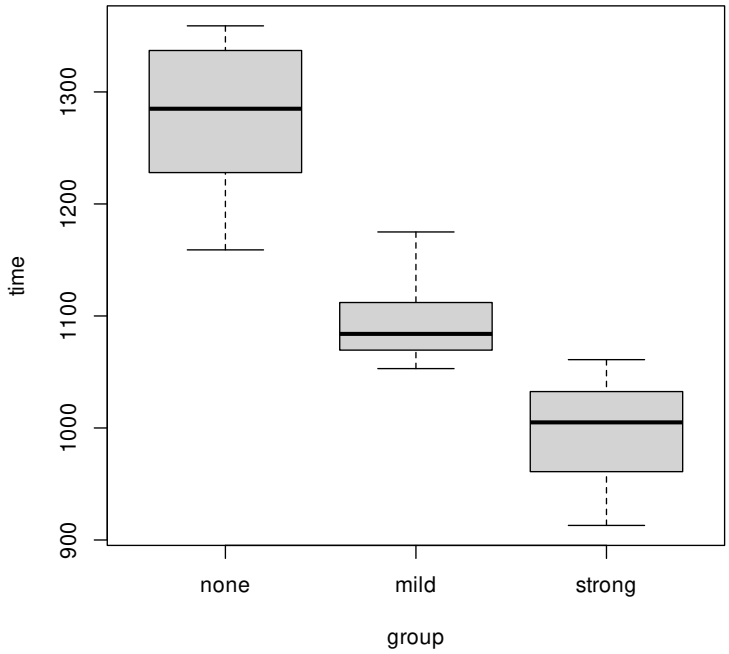

Figure 6.2: Quartiles of time for each experimental group; y-axis is truncated

We compute the means of each group:

```
aggregate(time ~ group, results, mean)
```

```
   group      time
1   none 1275.0000
2   mild 1097.0909
3 strong  993.5455
```

How much faster were the developers (i) with mild formatting compared to none and (ii) with strong formatting compared to none?

```
difference <- function(col1, col2) {
    diff = (mean(results[results$group == col1, "time"])
            / mean(results[results$group == col2, "time"]))
    paste(round(100 * (1 - diff), 3), "%")
}
```

```
print(difference("mild", "none"))
```

```
[1] "13.954 %"
```

```
print(difference("strong", "none"))
```

```
[1] "22.075 %"
```

To find an appropriate statistical test, we first determine whether the data are normally distributed. The distribution histograms for each group are shown in Figure 6.3.

```
par(mfrow = c(1, 3))

for (group in levels(results$group)) {
    hist(results[results$group == group, "time"], main = group,
         xlab = "time", ylab = "frequency")
}
```

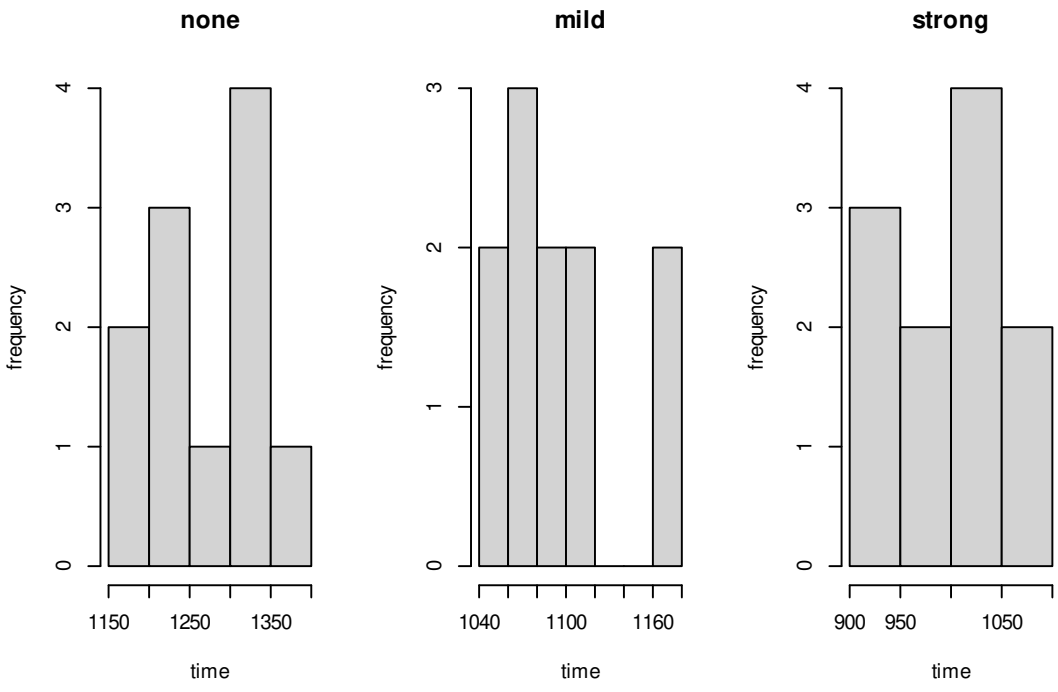

Figure 6.3: Distribution of time for each experimental group

Since the data are not normally distributed, we will use the Kruskal-Wallis test.

```
kruskal.test(time ~ group, results)
```

```
    Kruskal-Wallis rank sum test

data:  time by group
Kruskal-Wallis chi-squared = 27.312, df = 2, p-value = 1.173e-06
```

Since the p-value is less than 0.05, we reject the null hypothesis. There is a statistically significant difference between the groups.

6.8.6 Threats to Validity

- *Construct validity*: Timing may not be perfect (the subject can press the shortcut too late).
- *Internal validity*: Some developers might be far more experienced. However, this threat should be mitigated by randomization to a large extent.
- *External validity*: The source code and tasks may not represent the typical source code and location tasks used in practice. It is also questionable to what extent the Master's students represent developers in general.

6.8.7 Conclusion

Developers performed the best using the strong highlighting type (22% faster than with none), followed by the mild highlighting (14% faster than with none). The results are statistically significant.

Exercises

1. Describe a specific situation when performing a controlled experiment in computer science to show causation would be:

   a. unethical,
   b. too time-consuming or expensive,
   c. physically impossible.

   What would you do instead?
2. Which are independent and dependent variables in the following hypotheses and what are their scales?

   a. Computer science teachers with more years of experience spend more time weekly doing research.
   b. Using a terminal instead of a graphical user interface to launch the debugged application lowers the number of launches.
   c. On a scale from saddest to happiest, using IntelliJ IDEA makes Java developers happier than Visual Studio Code.

3. State a two-tailed hypothesis and both possible one-tailed hypotheses about an experiment comparing a touchpad and a trackball with respect to the precision of clicking on a target.
4. Name a hypothesis for which selecting a quasi-experiment would be most suitable.
5. Suppose we are designing a posttest-only controlled experiment. Describe a situation (other than the examples in this chapter) in which:

   a. a within-subject design would be perfectly usable,
   b. a between-subject design has to be used.

   Why?
6. Which statistical test would you use:

   a. for comparing two groups in a between-subject controlled experiment with a non-normal ratio dependent variable,
   b. for comparing three groups in a within-subject controlled experiment with an ordinal dependent variable?

7. Find a paper describing a controlled experiment. How does it categorize threats to validity? Name one from each category.
8. Describe an imaginary experiment ruined by a critical threat to its internal validity. How would you prevent it?
9. Find a paper, preferably in your research area, reporting on a controlled experiment. In its text, mark the null and alternative hypotheses, variables (along with their scales), type of the experimental design, and the used statistical test. How was the effect size reported? If some of this information is not explicitly stated, deduce it.
10. A experiment can also be performed without human participants – the "subjects" can be programs, files, executions, and so forth. Find an example of such a controlled experiment.

7 Human-Centered Methods

In this chapter, we provide an overview of common research methods that directly involve human subjects. This should not be considered an exhaustive list of strictly defined and clearly separated methods, but rather a selection of typical examples with certain characteristics that can be customized as necessary.

Many of the concepts and principles described here are also usable in purely computer-based studies. For instance, sampling also applies to projects in software repository mining studies, and qualitative coding (assigning tags to parts of text) can be performed on texts from an Internet forum.

7.1 Ethical Considerations

Before performing any empirical study with human participants, it is necessary to consider the ethical standpoint of research. Specific rules that apply depend on the institution and country where the research is performed, but there exist several general documents providing general guidelines.

One of them, the Belmont report specifies three basic ethical principles: respect for persons, beneficence, and justice. They imply a number of practical rules.

First, subjects have to know that they are participating in a research study, and their participation has to be *voluntary*. They should be informed about the general goal of the study. If the knowledge of specific details would affect the validity of research negatively, it can be postponed to the *debriefing* phase after the study. Deception, i.e., providing outright false information about the purpose of the study, should be used only if absolutely necessary and the true purpose must be revealed during debriefing. Apart from voluntariness, a person has to be able to withdraw from the study at any point.

Second, the benefits must outweigh the risks of the study. This applies both to the individual and society level. For instance, at the individual level, the participants in the study can receive small compensation, such as a voucher, or be informed about the results sooner than the general public if they wish so. At the society level, a slightly higher-risk study is justified if a significant breakthrough is expected thanks to the results, but the risks should not be higher than absolutely necessary and never cross a certain line.

Finally, the sample that will be studied should be selected fairly. For instance, if our research is aimed at professional software engineers, we should not constantly use only students in every experiment because they are conveniently available.

Before conducting research with human subjects, an *institutional review board* (IRB) approval is required in many cases. An institutional review board is a group of people at an institution, such as a university, who decide whether a given research study is ethical. An IRB application is institution-specific, but generally it contains information about the investigators, a hypothesis or research question, a succinct but complete description of the planned research method, and a sample informed consent document. An *informed consent* is an agreement of an individual to participate in a study. It is best if it has a written form (electronic or paper-based), since verbal informed consent is difficult to prove. An informed consent should contain:

- the name of the research project and the purpose of the study in simple terms,
- contact information of the researchers,
- the types of tasks that the subjects will perform,
- the data collection procedures and storage conditions,
- risks and benefits,
- and the confirmation that the person participates voluntarily and can withdraw at any time.

Research data obtained during research with human participants should be carefully scanned for personal information. All information that could reveal the identity of the participants should be anonymized (or pseudonymized) before publishing. Furthermore, relevant regulations such as the GDPR need to be taken into account.

7.2 Surveys

A survey, in this sense, is a method to collect and analyze data from humans using a questionnaire with a fixed set of questions. This questionnaire can be either paper-based or, very commonly, administered as a web form. The usual purpose is to find certain self-reported information about a population, e.g., the average number of monitors used by professional 3D modelers, or the most common problems Haskell developers face when using monads.

7.2.1 Sampling

While it is sometimes possible to administer a survey to the whole population, often this is not feasible. This is also the case for our example, i.e., the population of all Haskell developers in the world. In order to make a study practically executable, we

need to select a *sample* from the population. There are two principal ways to attain this: probability and non-probability sampling.

In *probability sampling*, every member (or item) of the population has a known, computable probability of being selected for the sample. For this, we ideally need a list of all members of the population. If it is not available, an imperfect *sampling frame* can be used, listing a large portion of the population. Two of the most common probability sampling strategies are:

- *Simple random sampling*: Each member has the same probability of being selected.
- *Stratified random sampling*: We divide the population into groups called strata. A *stratum* contains members having certain characteristics, such as junior or senior developers only. Then we can randomly sample members from each stratum. For instance, if there are 400 junior and 100 senior developers in the population, and we use a proportionate allocation strategy for a sample of 50 developers, then 40 junior and 10 senior developers will be in the sample.

When a list of all population members or a sampling frame is not available, we need to use *non-probability sampling*. Some of the commonly used strategies are:

- *Convenience sampling*: Subjects who are most conveniently accessible to researchers are used. This includes, for example, students in a teacher's class or employees in a company where the researcher used to work.
- *Self-selection* (*volunteer sampling*): Here, instead of the researchers sampling the subjects, the participants select themselves to become a part of the sample. For instance, after publishing a link to a questionnaire on a public forum, people who are interested can participate.
- *Snowball sampling*: After an initial group of people participates in the study, they recruit additional participants, which call more of them, etc.

7.2.2 Questions

Before designing questions in a survey, it is necessary to carefully consider what we would like to find out. Each survey question should contribute to answering one of our research questions or confirming our hypotheses.

Some RQs about simple variables and facts can be directly transformed into an item in a questionnaire. For example, if we want to know what IDEs Haskell programmers use, we can simply ask: "What IDE do you use?"

However, we often want to study abstract, multi-dimensional *constructs*, which cannot be directly measured or observed. For instance, asking "What is your cognitive load when merging Git commits?" does not make much sense. Constructs such as cognitive load need to be operationalized into measurable items. In the case of surveys, one multi-dimensional construct is usually operationalized in a set of multiple related questions,

either standardized or designed specifically for the given study by a researcher. In our example, the construct of cognitive load could be operationalized using the standardized NASA Task Load Index (TLX).

The questions asked in a survey can be categorized into two main groups: closed-ended and open-ended questions.

Closed-ended questions include numeric, single-choice (including the Likert scale), multiple-choice, item-sorting (ranking), and similar questions that can be directly analyzed quantitatively. They are useful to answer many kinds of RQs, mainly about counts, frequencies, proportions, such as: What is the average value of X? What proportion of X is Y? Considering multiple questions at once, we can also ask: Are X and Y correlated?

Open-ended questions require free-form texts, images, sounds or other unstructured data as answers. After qualitative analysis, they can be used to answer many kinds of "how", "why", "what", and other RQs, e.g.: What is the users' attitude to X? Why do people do X? How could X be improved?

7.2.3 Pilot Testing

Sending a survey to hundreds of subjects and then realizing the most important questions was ambiguous is really unpleasant. Before starting the actual response collection, first try the survey on a small group of people. This group does not have to be representative, and it is usually obtained by convenience sampling. Responses from the pilot testing will not be included in the final analysis.

During the pilot testing, we should focus on the clarity of the questions, appropriateness of the response scales, the time required to complete the survey, and technical issues. We can then modify the survey according to the feedback if necessary.

Pilot testing is often applied also to other research methods involving human participants, especially controlled experiments.

7.2.4 Preprocessing

If we used a sampling and recruitment method that involved invitations of particular persons, most likely not all of them actually filled in and submitted the questionnaire. It is thus necessary to mention the response rate in the report.

The questionnaire form should be configured to validate the inputs whenever possible, such as forbidding empty answers for mandatory questions. Nevertheless, the received responses should be checked for signs of invalid responses, such as free-form answers containing nonsensical texts or combinations of answers to two questions that do not make sense. There are three basic options to deal with invalid responses:

- Exclude the whole response of the given participant.
- Exclude only the invalid answer to the given question, but leave the rest of the answers of the participant as is.
- *Impute* missing or invalid values (Miao et al., 2023). This has to be done carefully so as not to lower the validity of the study.

Whatever the choice was, it has to be clearly documented in the report, including the specific counts of invalid responses.

7.2.5 Analysis of Results

The analysis and reporting of the results depend on the nature of the questions. Open-ended questions require specialized analyses, such as qualitative coding, which we will describe in the next section about interviews. Closed-ended questions are usually reported using descriptive or inferential statistics.

When choosing *descriptive statistics*, we can report means, medians, and standard deviations of percentages or absolute frequencies. With non-probability sampling, we should generally not claim representativeness for the whole population. For instance, you cannot say "40% of all C# developers use lambda expressions" if you surveyed only programmers from one company.

A large variety of *inferential statistics* can be computed from survey results. Some of the most common ones include the calculation of confidence intervals, differences between groups, and relationships between variables.

When the participants were selected using probability sampling, we typically report confidence intervals for individual variables. A *confidence interval* is a range of values that likely contains the true value, e.g.: "40% of subjects selected this option (95% confidence interval is +/-4%)". It consists of the definition of a confidence level and a margin of error. The *confidence level*, usually 95% or 99%, tells us that if we repeated the random sampling infinitely, the given percentage of intervals would contain the true value. The given interval is defined by the *margin of error*, e.g., +/-4% in our example.

Instead of post-hoc reporting of confidence intervals only to find that they are too broad because of an insufficient sample size, we can calculate the required sample size before starting the recruitment of participants. This can be accomplished with statistical libraries or an online sample size calculator for simple random sampling.

Differences between groups can be calculated when we have a categorical variable representing groups (e.g., the role: a manager, programmer, or tester) and another concept of interest (such as the number of lines of code written per day) captured in a survey. Then we can compute averages of the groups and determine whether the differences are statistically significant using an appropriate test, which is a procedure similar to the one

used in controlled experiments (Section 6.6). Of course, unless we include confounding factors in the survey and control for them, we cannot show causation.

Other *relationships between variables* captured in the survey can be computed too, such as *correlation* between two numeric variables. We will demonstrate this on a made-up example.

7.2.6 Example

We have a hypothesis that developers with more years of experience write more lines of code per day. Suppose we collected data using a survey.

A web questionnaire was designed, containing only two questions, both mandatory:

1. How many years of industrial experience do you have? (numeric answer)
2. How many lines of code do you write on average per day? (numeric answer)

Pilot testing with three subjects did not reveal any problems. Convenience sampling was used; an invitation to participate in the survey was sent to 30 developers in four software companies in one city. Two weeks were allocated for response collection. Twenty-one developers filled the questionnaire, resulting in a response rate of 70%. Two responses were excluded since they reported more than 100 years of experience. Results are shown in Figure 7.1 as a scatter plot.

```
results <- read.csv("data/survey.csv", header = TRUE)
plot(lines ~ experience, results,
     xlab = "experience (years)", ylab = "lines of code/day")
```

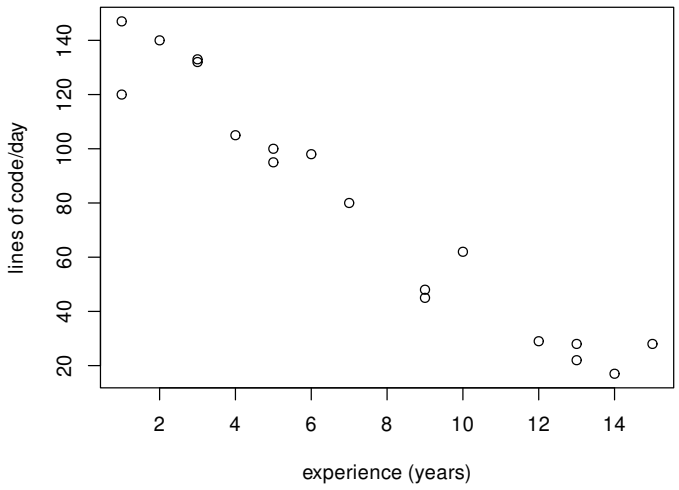

Figure 7.1: Correlation between developers' experience and lines of code per day

We calculate Spearman's ρ, which is a correlation coefficient usable for monotonic (including non-linear) relationships:

```
cor(results$lines, results$experience, method = "spearman")
```

```
[1] -0.9632526
```

The results indicate a strong negative correlation (-0.96). The hypothesis was not confirmed; on the contrary, according to our results, more experienced developers tend to write fewer lines of code per day. Our study is limited by an important construct validity threat, which is manual, approximate reporting of the lines of code written per day instead of automated measurement in an IDE. Obtaining responses from companies in one city is an external validity threat.

For more realistic and high-quality examples of survey research studies, we recommend papers by Inal et al. (2020) and Begel & Zimmermann (2014).

7.3 Interviews

An interview is a session focused on a specific topic when the researcher asks questions and the interviewee answers. An interview study is usually performed as a sequence of

multiple interviews with one researcher and one participant. An interview with multiple subjects at the same time is called a *focus group*. Typical RQs answered by interviews and focus groups are "how" and "why" questions: How do people perform some activity? Why do they perform it?

We know three main types of interviews:

- *Fully structured*: This is essentially a spoken questionnaire.
- *Semi-structured*: A researcher has a set of predefined questions, but additional questions can be asked.
- *Unstructured*: A researcher has a starting question, and then the dialogue continues freely.

Open-ended questions are often asked. The interview is often sound-recorded, with the participant's permission. In addition to or instead of recording, the researcher can take written notes. After the interviews, the recordings are transcribed.

7.3.1 Qualitative Coding

The transcripts are analyzed using a process called *qualitative coding*. It is a form of text analysis, applicable also for qualitative methods other than interviews. The researcher reads the text and labels selected segments with tags called *codes* that characterize each segment in some way. Let us consider the following text:

> When I connect to a meeting, I always forget to turn on the microphone. I usually talk for a minute straight until I realize it. …

A researcher usually starts with *open coding*, which is the assignment of initial codes without choosing from a predefined list or applying strict rules. For instance, we assign the code "forgetting turning on microphone" to the first sentence. To emphasize the interviewee's emotions, we could assign an *in vivo code* "talk for a minute straight", citing the used words exactly, to the second sentence.

After open coding, we could continue with *axial coding* that groups related codes into one. For example, "forgetting turning on microphone" and "forgetting turning on camera" could be both assigned the code "disabled input devices". Finally, we can perform *selective coding* by choosing a central topic and relating codes to this topic.

A more elaborate example by Graziotin et al. illustrates qualitative coding phases for studies about the happiness of software developers. Here, the codes are assigned by three researchers separately and then merged. As examples of papers using qualitative coding for interview data, we can mention the one by Krause et al. (2025) or Habiba et al. (2024).

7.3.2 Data Saturation

To determine how many participants are needed to maximize the validity of a study, quantitative studies use statistical techniques, such as power analysis. However, in qualitative research, such as interview studies, a completely different approach is necessary. Here we stop collecting data when *data saturation* is reached. Data saturation occurs when adding more participants does not provide new topics (e.g., codes in qualitative coding) anymore and information starts repeating to a great degree. Data saturation is discussed in detail by Saunders et al. (2018).

7.4 Diaries

Participants of surveys and interviews often supply imprecise data, particularly related to past events. This problem can be solved by using a data collection method called a *diary*. A participant performs tasks in natural settings and writes notes into a diary about selected events continually as they are occurring. Diaries alone or with a connection to other data collection methods help answer RQs such as: How does X occur? How often does X occur? What does a person X typically do?

A participant can write a record to the diary:

- Each time a specific event occurs: e.g., "Each time when a program crashes, write what you tried to accomplish and how it affected you."
- Periodically: e.g., "Every two hours, write what problem you are working on."

7.5 Observation Studies

In *observation studies*, the researcher analyzes the actions performed by participants during tasks. Specific details depend upon the nature of the task. If the goal is to assess the usability or other properties of some software, the screen contents are often captured in the form of a video. This could answer the "how" questions perfectly, but to find out why certain actions were performed, we should complement it with a *think-aloud protocol*. Here, in addition to performing the task, the participant also comments on the reasons, intentions, and other ideas behind the actions.

Studies aiming at understanding cognitive load, physical effort or similar characteristics, can utilize *biometric measurement* in addition to the participants' observation. Eye tracking, movement and position tracking, and brain activity monitoring using EEG (electroencephalography) or fNIRS (functional near-infrared spectroscopy) pertain to this category.

A particularly interesting technique called the *Wizard of Oz* allows us to perform an observation study with a system or approach which has not yet been developed. Instead of a computer, a human responds to the participant's input, simulating the output that we would expect from the system if it had been implemented. It is useful to obtain qualitative data in the early exploratory phases of a research project.

7.6 Methodological Approaches

Multiple methods and data collection or analysis techniques can be combined to form a larger methodological approach. We will mention a few of them in this section.

A *case study* is an in-depth investigation of a small number of cases in their natural context. A *case* is a unit of analysis, e.g., a project, team, organization, or technology, bounded by some criteria, such as time, place, or other properties. Case studies use multiple data sources and predominantly qualitative analysis. They are not generalizable in a statistical sense but nevertheless can provide interesting insights about a phenomenon, transferable to other contexts. There are multiple types of case studies. For example, an *exploratory case study* could be used to find which challenges graphical user interface designers face. We would visit one or two companies, observe them for some time and take notes. Then we would interview them about their problems. In a *demonstration case study*, we could offer your newly designed tool to a small group of developers in one organization to use it for real-world work. Then we would ask them for feedback.

Ethnography is a combination of observation, interviews and typically also active participation of the researcher with the study subjects directly in their natural settings. *Phenomenology* studies the lived experiences of participants, focusing on a certain phenomenon and usually without active participation of a researcher. Finally, a *grounded theory* uses a variety of methods and data sources to build a new theory from data.

Exercises

1. In what research study with human participants in computer science would deception be necessary for validity?
2. Is it ethical if students receive course credit (points) for participating in research studies? Why or why not?
3. Find a paper in your research area describing a survey (questionnaire research). Was the population or the sampling frame explicitly mentioned? Which sampling strategy was used?
4. Find a computer science paper using stratified random sampling – not necessarily for a survey. What were the strata?

5. Some researchers claim that everything, including simple variables such as the age of a person or the number of lines in a file, is a construct that cannot be measured directly and thus needs operationalization. What arguments support their claim?
6. What specific problems can be discovered during the pilot testing of a research study? You can answer using your own experience, an existing paper, or a made-up situation.
7. What are advantages and disadvantages of excluding only invalid answers to specific questions in a response instead of the exclusion of the whole response of a participant?
8. If you had a list of all 12,000 users of the library FooBar, how many participants would you need for a survey to claim that a certain percentage of the users wish FooBar was implemented also for Lua with a 2% margin of error and 95% confidence interval?
9. Select an excerpt of direct speech text related to computer science, about ten sentences long. Try to perform open coding and then axial coding on it.
10. Find a paper using

    a. a think-aloud protocol,
    b. biometric measurement

    in some subfield of computer science that you like. What research question was answered specifically thanks to this method?
11. How specifically would you use the Wizard of Oz technique?
12. Find a paper describing a case study. What exactly was the case and how was it bounded? Were there multiple cases?

8 Computer-Based Methods

After presenting research methods with human participants, we continue with methods that are either executed on computer systems alone (benchmarking, archival data analysis, and simulation) or reason about computer behavior (proofs). Again, this is not an exhaustive list, and the individual techniques can be mixed and matched according to the specific needs.

8.1 Benchmarking

In many subfields of computer science, we need to compare multiple systems or approaches according to some criterion. This is commonly accomplished with a benchmark study.

A *benchmark* is a standardized test (a program or a dataset), combined with a prescribed usage procedure, whose main purpose is to evaluate and compare systems with respect to their performance, reliability, or other characteristics. Based on a set of inputs, operations, and conditions called *workload*, it produces *metrics*, such as execution time or F1 score.

The main advantage is an easy and fair comparison of multiple systems, which fosters competition and advances research on the given topic. A disadvantage is that researchers can start optimizing their approaches toward the benchmark, without considering practical applicability.

According to Gray (1992) and Huppler (2009), a good benchmark needs to be:

- Relevant: It should measure the performance of typical real-world operations and be of practical importance.
- Repeatable: We should be able to reproduce the results by running the benchmark again under similar conditions.
- Portable: It should be easy to port the benchmark to other systems or architectures.
- Scalable: A benchmark should be executable with a smaller/lighter workload to be economical enough, but also with a larger one for realism and precision.

Kounev et al. (2025) divide benchmarks into specification-based, kit-based, and hybrid. *Specification-based* benchmarks provide natural-language description of the workload, while the implementation is up to the user. This allows for greater customizability, but reproducibility is challenging. *Kit-based* benchmarks provide a complete implementation and a *hybrid* benchmark only a partial one.

8.1.1 Application

There are two common ways to apply benchmarking in a research study. The first way answers RQs such as "Is X faster/more efficient than Y?" or "Which approach is the best for X?". Here, we compare a target approach (often the one proposed in the paper) against a baseline or, even better, multiple baselines. A *baseline* is a relevant competitor of the target approach. Baselines can include a naive algorithm or a commonly used approach in practice. However, the most important baselines are *state-of-the-art* approaches, which are the currently best ones, according to the specified criteria, published in research literature. Even if a state-of-the-art approach is not applicable to a formal benchmark at this time due to some technical limitations, we should try to compare the target approach with it to the maximum extent possible.

Second, we can compare multiple variants, parameters, or input sizes for one approach. For instance, an algorithm has two variants, non-cached and cached, where the cache size is configurable in kilobytes. We can plot the non-cached variant and various cache sizes against performance. This could answer questions such as "How does X scale?" or "What are the optimal values of X?".

8.1.2 Examples

The details of benchmark application vary greatly across computer science subfields and the specific metrics that the benchmark produces. As an example, we will explain time-based performance measurement of programs. Suppose we would like to measure how various compiler options affect the speed of a running program. After compiling the benchmark with given command-line options, we execute it on an otherwise idle machine. Possible metrics that the benchmark could output include:

- *real time* (wall clock time): the total execution time, including input/output operations and waiting,
- *user* and *system time*: time spent executing processor instructions in the user mode and the operating system's kernel mode, respectively,
- *CPU time*: a sum of the user and system time,
- and *throughput*: the number of operations (e.g., mathematical computations or data items processed) per second.

Often the benchmark can and should be configured to run the workload multiple times to get statistically more stable. Then, from these basic metrics, a median, mean, standard deviation, and percentiles can be computed. Some benchmarks, especially for just-in-time compiled runtimes, include *warmup rounds* that are excluded from results.

Configuring the benchmark for multiple executions is crucial when one execution takes too little time, such as a few milliseconds. Short executions would be dominated by noise in the measurement coming from the operating system process switches, imprecise

timers, and initialization of caches. Note that the benchmark itself should support this configuration; simply running the whole program separately multiple time typically does not solve this problem.

As specific examples of papers describing the development of a new benchmark, we can mention the DaCapo benchmark suite for Java performance (Blackburn et al., 2006) and BBEH for natural language reasoning tasks (Kazemi et al., 2025).

8.2 Archival Data Analysis

Instead of performing research studies directly with human participants, we can resort to the analysis of existing artifacts that people produced without the intention of a participation in research. This is often less resource-intensive and offers a look into natural settings instead of full control. Similarly to studies performed with humans, we should not forget to anonymize any personal information that we encounter.

Archival data analysis is applicable in numerous computer science subfields and has many different forms. We selected two specific examples: mining software repositories (MSR) and the analysis of discussion forums.

8.2.1 Software Repository Mining

Software repository mining is a method in the area of software engineering that aims to answer research questions by systematic analysis of data present in version control systems, issue trackers, mailing lists, and similar systems. It is useful to answer a wide range of questions, from "How to predict which commit introduces a bug?" (X. Yang et al., 2015) to "Do programmers work at night or during weekends?" (Claes et al., 2018).

After posing research question(s), we choose a repository or a set of repositories that contains data suitable for answering it. This is often a large open-source project hosting website such as GitHub, which offers an integrated issue tracking and continuous integration features, but it can also be a collection of smaller, organization-specific repositories (e.g., Apache or Mozilla) interconnected with external systems. Unfortunately, MSR tends to have a deficit of industrial studies due to intellectual property problems.

Then, we clearly specify inclusion and exclusion criteria. For instance, we can limit the repositories to certain programming languages, libraries used, last update dates, and other criteria according to the purpose of the study. Sampling (Section 7.2.1) may be necessary if the set of projects is too large. If using GitHub, excluding noise in the form of toy projects, personal backups, or homework assignments is also advised (Munaiah et al., 2017).

Data downloading and extraction follows. We recommend reserving enough time for this phase, as software hosting sites often impose API rate limitations. The downloaded data

should be saved in standard, platform-independent formats. A subset of data necessary for the purpose of the study is extracted and processed. The nature of processing depends on the nature of the study and can include descriptive statistics, correlational analysis, machine learning, and even manual analysis of parts of the data.

Vidoni (2022) provides a systematic review of MSR studies, along with a generic description of the process most studies use. Kalliamvakou et al. (2014) list nine facts that should be taken into account when doing a MSR study using GitHub to prevent significant threats to validity. For instance, many projects are inactive, have very few commits, or do not use the GitHub's pull request feature.

8.2.2 Analysis of Discussion Forums

Another common method utilizing data from humans without involving them in a research study is the analysis of Internet forums, Q&A (question and answers) websites, and social networks.

A typical method begins with posing research questions and selecting appropriate websites. Next, we decide which kinds of data we need: posts, likes/votes, creation dates, authorship relationships, etc. We continue with sampling of the given posts, using inclusion and exclusion criteria that we define. After data extraction, it is necessary to preprocess the the data, as particularly the text of posts often contains formatting marks or has character encoding issues. The analysis can be performed in a quantitative, qualitative, or mixed methods manner. For instance, the text of the posts can be qualitatively coded, similarly to the case of interviews (Section 7.3.1). A social network can be modeled as a directed or undirected graph and quantitatively analyzed using graph algorithms (Tabassum et al., 2018).

In the area of software engineering, hundreds of papers using the Stack Exchange network, particularly the Stack Overflow Q&A website, were published. They range from general analysis of topics – "What topics are mainly discussed on Stack Overflow?" (Barua et al., 2014) to the comparison of ChatGPT answers' quality with Stack Overflow (Kabir et al., 2024). The analysis of Q&A websites is often combined the software repository mining (D. Yang et al., 2017).

8.3 Simulation

Computer *simulation* is a method consisting of the development of a mathematical model and its subsequent execution to derive results. In other words, it is an approximate execution of a process in a contrived setting.

Simulation is particularly useful when the real executions would be prohibitively expensive, time-consuming, unethical, or difficult to observe. It can be used to answer many "what-if"

RQs, such as "What if X happened?" or "How does X probably behave under condition Y?"

Computer simulation is used in a multitude of research fields; human-computer interaction is among the more interesting ones. Some papers focus on specific applications, such as the one by Dubey et al. (2021), which describes a simulation model for wayfinding using signage in 3D space. On the other hand, Park et al. (2023) devised a general simulation of a village with 25 agents performing human-like behavior.

8.4 Proofs

Finally, we briefly mention mathematical proofs, pertaining to non-empirical methods, for completeness. By a *proof*, we understand a rigorous, logical sequence of mathematical statements showing that a certain statement is true.

Generally, the research questions that proofs aim to answer are of type "Does X exist?" or "Does X hold for all Ys?" Formal proofs are heavily used in theoretical computer science. Other areas relying on proofs are programming language theory, cryptography, and algorithm design.

Proofs are useful alone, but sometimes they are combined with empirical methods. For instance, when proposing an algorithm, its time and space complexity can be proved mathematically and then measured empirically on a range of inputs to show how it behaves in reality, taking hardware peculiarities into account.

Lamport (2012) provides useful tips on how to write readable and more understandable proofs. According to him, we should break them into hierarchical structures, with high-level proof sketches first and details deferred to a later part. A machine-checkable version should also be supplied if possible.

Exercises

1. Discuss a real or potential situation where the researchers in some computer science subfield started concentrating too much on beating a benchmark, while factors limiting the application in practice were neglected.
2. Find at least two papers that use the same benchmark in their evaluation. Was the benchmark itself published in another paper? Are the evaluation results directly comparable, or are there discrepancies (e.g., different metrics or conditions)?
3. Find a study that mines software repositories or analyzes discussion forums. What were the inclusion and exclusion criteria for repositories/posts? Were all matching repositories/posts analyzed or was sampling used?

4. Suppose you prove that an algorithm has an average-case time complexity of $O(n^3)$, but after implementing it, executing on real data, and plotting the results, you find out the chart seems more like $O(n^2)$. What could be the reasons?

9 Writing

This chapter provides guidelines and tips on writing various types of documents reporting research. We will focus specifically on research papers aimed at publication in a journal or conference proceedings, but the majority of the tips apply also to master's and PhD theses, as we mention later.

9.1 Paper Structure

Research papers tend to have a standardized structure that helps readers find information they are looking for quickly. In general, the research paper consists of a title, abstract, introduction, main paper body (including the methods and results), related work, and conclusion.

Specific, detailed structure depends on the type of the paper and the research methods used. When in doubt about the structure of our paper, we can use high-quality papers in the given area describing a similar research method for inspiration. For example, if we are conducting an unstructured interview in human-computer interaction, we can use a high-quality HCI paper describing an unstructured interview as a reference, even when the topic is different. For specific situations, there may even exist guidelines, such as the Reporting guidelines for controlled experiments in software engineering (Jedlitschka & Pfahl, 2005) and others.

The section structuring must be logical; the relationship of a subsection to its parent section should be "is-part-of", "is-type-of", "is-aspect-of" or similar. When describing multiple entities, e.g., tools, methods, or experiments, order them in some consistent way: chronologically, by specificity, by efficiency, forming a data flow from inputs to results, etc.

Similar to reading, a paper is usually not written head-to-tail. Starting with a hierarchical outline consisting of section headings takes little time but provides a quick overview for self-critique and collaboration. We should reorder or change section structure during writing whenever there is a need.

An overview of the standard parts of research papers follows.

9.1.1 Title

A *title* should be as short as possible to uniquely identify the paper's main contribution, but not shorter. A rule of thumb is that there should always exist only one published paper with a given title in the world. Naming a paper "Tree-Based Database Index" would be too general nowadays, as it suggests this is the first paper proposing tree-based database indexes ever. On the other hand, the title "Design, Implementation, and Evaluation of Extremely Time- and Space-Efficient Indexes in Relational Databases Based on A Combination of AVL Trees and B-Trees" contains superfluous words.

9.1.2 Abstract

An *abstract* is a very compressed summary of the whole paper. Since a reader will often decide whether to read your paper or not based on the abstract, it should receive appropriate attention. In general, the abstract should contain:

- Background (context): Important facts that are necessary to understand the research problem.
- Problem statement or goal: What are we trying to solve or find out and why?
- Approach and/or method: A very high-level overview of the designed approach (if any) and the research methods used.
- Results: Specific (e.g., numeric) main results found by applying the method described above.
- Conclusion: The implications that our solution has in a larger context; possibly also limitations or future work.

Ideally, each part should be approximately the same length. A common beginner's mistake is a too long background description, mentioning facts that almost all researchers in the area already know.

The present and past tense should be used in the abstract, unless an actual future event is described. An abstract should not contain too general statements; e.g., "we discuss the results and review related work" applies to almost all papers. An abstract is usually written as one paragraph, but some journals encourage or require structured abstracts, containing named inline headings.

In Koopman's essay on how to write an abstract, we can find more specific useful hints.

9.1.3 Introduction

An *introduction* is a more verbose summary of the paper with less or no details about the method and specific results. Instead, a special emphasis is on the *motivation*: What is the problem and why is it worth solving? Note that motivation cannot be relevant solely

to the authors of the paper, such as "we are interested in this topic" or "our department has a spare box of Raspberry Pis in a storage room."

According to Booth et al. (2016), an introduction should consist of three main parts: context, problem, and response. *Context* is information that the readers of the paper probably already know, relevant to our main goal. It sets the scene for a *problem*, which names the shortcoming(s) of the current state of the art, such as missing parts of knowledge or disadvantages of existing techniques. After reading the problem description, the motivation should be obvious for the reader: Do we gain a certain advantage by pursuing this research? Or will ignoring the problem have unintended consequences? A *response* is our solution to the problem, which is often an empirical study aiming to fill a knowledge gap, a design of an improved approach, or both.

A typical (significantly shortened) example of this three-point structure may look like this:

> Smartphone users rely heavily on touch keyboards to enter textual input. (context) However, touch keyboards are significantly less accurate than the physical ones. (problem) Foo et al. [1] measured a 54% lower accuracy, while Bar et al. [2] even a 67% lower one. (problem evidence) Haptic feedback improves accuracy only marginally [3]. (problem of existing solutions) We present a 3D keyboard temporarily raising over the display surface, which provides user experience similar to physical keyboards. It was successfully validated in a controlled experiment with 40 participants. (solution)

In the introduction, we cite relevant sources, but not all of them. We should focus there only on the most significant references that directly cause, acknowledge, or try solving our problem. Although we describe our solution briefly in the introduction, detailed and specific results are usually left until the later parts of the paper.

A bulleted list of contributions (e.g., design of a new method X, a formal proof of the algorithm Y, and a survey on topic Z) is sometimes appreciated by reviewers. Some reviewers also require a paragraph resembling a table of contents (In section 2, we review related work. Section 3 describes …), but others despise it.

When describing the problem we try to solve in the paper, we can illustrate it on a specific example, e.g.: "Suppose a programmer searches for all occurrences of the method xyz()…" Then we can optionally continue by following up on it in the rest of the paper, which is called a *running example*.

9.1.4 Main Paper Body

Which sections follow after the introduction differs widely between papers. For empirical studies, the generally accepted section structure is:

- Method,
- Results,
- Discussion,
- and Threats to Validity.

The *Method* section describes the design and procedure of the empirical study that was performed. The method should be described in sufficient detail to prevent ambiguity as much as possible. The *Results* section should clearly and objectively describe the results obtained using the already explained method. In the *discussion*, we can discuss more subjective or interpretative questions such as: What is the cause of the given results? What implications do the results have? Why are they important? In some cases, particularly if the discussion would be too short, we can merge it with the *Results* section, provided we distinguish between objective results and our interpretation. Finally, the *threats to validity* discuss in what ways our results might not be valid and what we did to mitigate them if possible.

Papers describing artifacts (design science), i.e., new approaches, methods, prototypes, or systems, do not have any prescribed structure. The general rule of appropriate section ordering still holds though. We often start with an overall high-level design and then describe the individual smaller components, e.g., in the order of the data flow. The order in which the author designed and implemented the parts is generally irrelevant and should not be mentioned unless it is critical for understanding.

In any case, the section structure is not carved in stone. For example, a paper can contain multiple experiments, each described in a section with its separate Method and Results subsections. If a paper also contains a design of a new approach, we can, for instance, describe a new algorithm in the section “Indexing algorithm” and then evaluate it in the section ”Evaluation” with the subsections Method and Results.

Since an implementation of a prototype is a time-consuming activity, many beginning computer science researchers think that implementation details deserve to be described thoroughly. On the contrary, we should refrain from including source code excerpts unless they represent the core idea. Even then, pseudocode is better. Mentioning the specific classes, methods and variables representing implementation details should generally be avoided in the body of the paper. Does the reader really need to know that the result of calling `getCoordinates()` is assigned to `currentCoordinates`?

9.1.5 Related Work

In the *Related Work* section, we reference literature as relevant as possible to the research that we performed. We should try to find papers related to our research from various aspects. For example, if we performed a survey about agile practices in smartwatch apps development and then designed an issue tracker based on the findings, we should mention both empirical studies of the exact same topic and existing issue tracking approaches

aiming to solve the problems that we found. If, however, after an honestly exhaustive search we find no papers discussing agile practices in smartwatch apps development, we can broaden the scope a bit, e.g., to agile practices for mobile devices in general.

For each reference, we briefly describe:

- its synopsis (main findings),
- how it is similar to our research,
- and how it is different.

The Related Work section can be placed after Introduction or before Conclusion. The latter is preferred if it does not impede understandability.

Searching and describing related work should start early, but then continue incrementally as we progress with the paper, having a clearer understanding of some aspects. As we mentioned in Section 3.5, processing related work is much easier with a reference manager.

9.1.6 Conclusion

The *Conclusion* section is, again, a summary of the article, but this time leaving out the motivation and focused on the results instead. In the conclusion, we describe:

- the main point and the most important results of the paper,
- applications and implications: how the results can be used, even from a broader point of view,
- the limitations of our research,
- and possible future work.

9.2 General Guidelines

In general, we should write our paper in a way the reader or reviewer:

- will understand it,
- and will be convinced the research is relevant and valid.

Therefore, the first question that we need to ask is: *Who is our reader?* The readers of research papers are usually researchers working in the same subfield of computer science (e.g., operating systems or human-computer interaction) who were, sadly, not looking over our shoulder when we actually did the research. There is thus no need to explain how Git works in a study of GitHub projects. On the other hand, a random software engineering researcher has no idea whether we sampled the repositories using simple random sampling and how exactly did we even enumerate a list of all repositories (as GitHub itself does not have a time-efficient feature for this).

While a paper seems like a monologue, it is in fact an imaginary conversation with a skeptical reader (Booth et al., 2016), who we try to *convince* about the relevance and validity of our paper using *arguments*. When we are writing, we should imagine we are the reader (or better, the reviewer), try to anticipate relevant questions, and answer them at appropriate places. A good way to achieve this is to form research arguments in a structure according to Booth et al. (2016). An argument can contain a claim, reasons, evidence, the acknowledgment of objections with responses to them, and a warrant.

A *claim* is a statement that is true according to us, and we try to persuade the reader about its truthfulness by providing logical *reasons*. A reason should be supported by *evidence*, such as the results of empirical research or a formal proof. We can then *acknowledge* most relevant potential *objections* of a reader: Why the reasons we mentioned could be invalid or irrelevant and the evidence insufficient? We *respond* to such objections by providing further claims, reasons, and evidence. In rare cases, a *warrant* may be necessary to explain our way of thinking, i.e., why given evidence supports the reason, which, in turn, supports the claim.

An argument can be illustrated on this condensed example:

> Absent Readme files are an impassable obstacle for newcomers of open-source Java projects, $_{\text{(reason)}}$ as 97% of respondents would not attempt to build a project without Readme $_{\text{(evidence)}}$. An automated Readme generator could thus offer faster onboarding of newcomers to open-source projects. $_{\text{(claim)}}$ Although there are other obstacles for open-source involvement besides the failure to build the project from source $_{\text{(acknowledgment)}}$, this reason was by far the most frequent in the survey. $_{\text{(response)}}$

In the example, a warrant could be that building a project from source is a necessary precondition to become a meaningful code contributor in a Java project – but the readers probably already know that, so we did not explicitly mention it.

We can write great arguments, but if a reader does not *understand* what we write, our effort is futile.

Research papers are often limited in length, either strictly by a page count or loosely by the time a reader is willing to allocate to reading it. In a too long text, the main point can be lost easily. We should aim to write in the shortest way possible while still preserving unambiguity and understandability. Ideally, in a paper, "every sentence should be necessary" (Zobel, 2014). To achieve this, we should choose a central point (or a few points) of the paper – a message that we would like to deliver to our reader. Then, we imagine dependencies of this point, which are sentences logically leading to it, such as reasons, evidence, or other statements that support or explain it in some way. We repeat this transitively. Finally, everything that is not a transitive dependency of the main point(s) should be deleted. This rule cannot, of course, be applied literally. However, if in a paper on the automated generation of user interfaces for Android, we read that

Android Inc. was founded by Andy Rubin and colleagues in 2003, we know something is definitely wrong.

Some (especially inexperienced) researchers try to hide the fact that not much work was actually done by deliberately obfuscating the paper. We should write clearly and be honest about how the research was performed. Obfuscation will only make the review more difficult, not more positive.

In The Craft of Research, Booth et al. (2016) provide a few useful stylistic suggestions on how to write easy-to-understand texts:

- Denote actions in the text by actual verbs; avoid complex subjects produced by nominalization. For example, "A participant clicked the button five times because the user interface of the calendar was unresponsive" is better than "Due to the unresponsiveness of the user interface of the calendar, a participant clicked the button five times."
- Start a sentence with simple concepts or terms already mentioned in the text; new or difficult concepts should be moved to the end. For instance, "Two patterns are combined into a multi-pattern. This multi-pattern is then processed using meta-parsification by the ultra-foobarificator" reads better than "A multi-pattern is produced from two patterns. Using meta-parsification, the ultra-foobarificator then processes this multi-pattern."
- Do not hesitate to use active voice ("We showed that…"), but use passive voice ("The result was produced…") when it fits better semantically or to satisfy the two previous rules.

For more guidelines on how to write a great research paper, we recommend a presentation by Simon Peyton Jones (it also has a shorter version).

9.3 Writing Process

Writing a paper or a thesis is not a linear process. People rarely produce perfect writing on the first try. We should start writing a rough draft, noting our ideas as they come, even using unfinished sentences. Then we can return to individual portions of the paper and modify, reorganize, and delete them as necessary. Thus, unless the paper is already sent for review, we can conclude:

> A research paper is not a blockchain, we can modify it!

During writing, we should read the paper regularly ourselves and ask whether it would make sense for our readers. Before submission, we need to read it completely from start to end to find consistency issues and fix stylistic and grammar errors.

9.4 Tables and Figures

Each table and figure needs to be appropriately labeled. The label should be specific enough, e.g., "Correlation between age and the number of source code lines written per day" instead of "Age and lines".

Columns of a *table* should be named. If it makes sense, rows can be named too. Horizontal lines in tables should be used as prescribed by the given template, while vertical lines are to be avoided completely (unless the template says otherwise).

To graphically represent data, many researchers choose inappropriate *chart* types. For instance, they overuse pie charts, which make it difficult to visually compare the quantities. The type of a chart needs to be chosen based on what we need to show to a reader, e.g., for a comparison of cyclical data with many periods over time, we can use a circular area chart. To select an appropriate visualization, we recommend the document Chart Suggestions – A Thought-Starter by Abela.

Each dimension in a plot (x- and y-axis, color, shape) must be labeled so that the reader knows what it represents. Truncated (not starting at zero) or exponential scale can be used only if it is really necessary, and even then, it has to be explicitly mentioned in the text describing the chart or in its label. Visual elements should have meaning; they should not be added for the sake of decoration, such as in the case of 3D pie charts.

9.5 Citing Sources

In a research paper, we need to cite sources in a way that clearly distinguishes these three options:

- our own idea (no reference),
- a paraphrased idea: according to X [1], something is true,
- and a quoted phrase: according to X [1], "something is true."

Although BibTeX generates the reference list at the end of the paper automatically, we need to check that the entries are valid and complete. Some bibliography styles transform all letters of a title except the first one to lowercase. This can lead to errors such as "The java and jvm specification", which can be fixed with braces in the *.bib file: `The {J}ava and {JVM} Specification`.

Some bibliography styles pair with the natbib package and author-date citations that produce natural-sounding sentences, e.g.: "Smith (2020) described…". However, even when the publisher's template prescribes numeric citations (and recommends not using `natbib`), we would often like to mention the authors' names in sentences for fluency. Then we need to write the names manually according to these rules:

- for one author, mention the surname only: Smith [1],
- for two authors, connect the surnames with "and": Smith and Novak [1],
- and for three or more authors, use "et al.": Smith et al. [1].

9.6 Writing Theses

After reviewing the rules and guidelines common to many types of research documents, including both papers and theses, in this section we focus on the specifics of writing master's and PhD theses.

9.6.1 Master's Theses

At this place, we could expect a long list of elaborate tips on writing theses. Instead, there is only one tip:

> A master's thesis should ideally be a longer version of a research paper.

We should thus aim to write the *master's thesis* using exactly the same rules and hints, with a few minor differences, such as:

- Related work is mandatorily placed after the Introduction, and it can (and should) consist of multiple chapters named appropriately, i.e., not simply "Related Work". It can be more verbose and detailed, but it cannot become "Unrelated Work".
- Technical details, such as architecture diagrams or measurement instruments (e.g., survey questions) can be described more thoroughly. Still, we should not go into a level of detail that is insignificant for the reader.
- Compared a research paper, a thesis also aims to demonstrate that the student is capable of doing research. Given more space allocated for writing, we can state more reasons, evidence, acknowledgments, responses, and warrants in our arguments.

Of course, this is an ideal situation that may or may not be attainable, depending on the specific topic of the thesis and the author's ambitions.

9.6.2 PhD Theses

Usually, a PhD student has multiple papers published during the studies. A *PhD thesis* is then often a collection of multiple research papers written by a PhD student on the same topic. For instance, each chapter is based on one research paper: first a systematic literature review, then a survey of the problems, then a solution to the problem with a preliminary evaluation, and finally a full validation of the solution.

Besides stapling the papers together, we need to write the Introduction and Conclusion chapters that clearly describe the logical connection between the individual chapters or papers. Then, the content of the chapters should be re-read and edited, so that the thesis is readable from start to end and makes sense as a whole.

Exercises

1. Shorten the long title mentioned in Section 9.1.1 appropriately. You can list multiple variants, along with corresponding hypothetical assumptions that make a given variant the best choice.
2. Find one good abstract and one abstract that breaks multiple recommendations.
3. Mark the context, problem, and response in the Introduction chapter of your bachelor's or master's thesis.
4. Find a paper that includes source code snippets in its main text (not appendices). Is it well-justified in this case or only a sign of low quality?
5. How does a conclusion differ from the introduction? You can illustrate this on a hypothetical example.
6. Extract a complete section structure from a conference paper about 8–12 pages long. Is it logical? How deeply is it nested?
7. In a paper that you like, find three or four most relevant related works (as indicated by the authors). In which sections are they present? Are they referenced multiple times?
8. In an existing paper, find an argument consisting of a claim, reason, evidence, acknowledgment/response, and optionally a warrant.
9. Write a short text (2–4 sentences) that satisfies the first two stylistic rules from The Craft of Research mentioned at the end of Section 9.2. Rewrite the text so that it violates them.
10. What type of chart would you use to show:
    a. the correlation between the monitor resolution and the average number of applications open simultaneously,
    b. the average resolution of monitors from year 1970 to 2025?
11. Which of the rules and tips in this chapter that you did not know before will help you improve your master's thesis the most? Do you see any impediments why you will not be able to apply some of them?

10 Reviewing and Publishing

After we finish writing a paper, we send it for review to an appropriate *publication venue*, usually a journal or a conference. If the outcome of the review process is eventually positive, it is published in the given venue.

10.1 Authorship

Although a collective that writes a paper usually decides on authorship sooner, when sending the paper for review, they need to make a final agreement about the exact list of the authors because it cannot be changed later. Each author should, according to the IEEE guidelines, fulfill all three criteria:

- intellectually contribute to the content (system design, method, results, etc.),
- draft or revise the text of the paper,
- and approve its final version.

In computer science, author names tend to be ordered by the significance of the contribution, starting with the most significant, but this is not enforced in any way.

Some journals require specifying contributor roles according to the Contributor Role Taxonomy (CRediT) in each article. CRediT defines fourteen roles, such as "conceptualization", "data curation", and "software". The authors append the specification of their roles to the text of the paper, for example:

> Conceptualization: J. Doe, K. Foe; Data curation: K. Foe; …

10.2 Choosing a Publication Venue

Many publication venues exist, differing in topics, acceptable contribution types, quality, and review process. We should always read the instructions on the website of the given venue to decide whether our paper fits it. Reading a few papers published in the venue gives us a feeling of what will be expected from us. In fact, it is best to submit to a venue whose papers we often read, provided the quality of our work suffices.

The two most common types of publication venues are journals and conferences. Computer science *journals* are typically suitable for mature research results with complete evaluation.

On the other hand, many *conferences* accept also works with preliminary evaluation, particularly in their Early Research Achievement (ERA), New Ideas, "short papers", or similar *tracks*, poster sessions, and co-located *workshops* or working conferences. In computer science, many researchers publish a paper at a conference first, and then they submit a significantly extended version to a journal, taking into account the rules of the publishers on what "significantly extended" means.

Other types of publications are non-conference proceedings and books. *Non-conference proceedings* are collections of papers, usually published annually, that do not involve presenting at a conference. In general, their quality criteria tend to be less strict than that of journals and conferences. *Books* in computer science are traditionally used to summarize highly developed topics, only rarely for original primary research.

Generally, we cannot publish the exact same content twice, which makes sense. Copying large portions from one of our published articles to another is also forbidden, unless we clearly claim what is the original contribution of the new paper with respect to the old one (and both publishers' copyright rules allow it). However, bachelor's, master's, and doctoral theses submitted to a university library are *non-archival publications*, i.e., they are not considered formal academic publications per se. This practically means we should be allowed to publish their large portions, such as chapters, as original research papers in a journal or conference. Other non-archival documents include *technical reports*, which are simply articles uploaded on the web, lacking any review process.

When selecting a publication venue, one of the criteria we could take into account is its *acceptance rate*. It is defined as the number of accepted papers divided by the number of submitted papers, as a percentage. Rather shocking for newcomers, the acceptance rate of top-ranked conferences and journals can be as low as 10 to 20%. Some journals display acceptance rates on their publisher's website, but many do not. For selected conferences, the Computer Science Conference Statistics shows acceptance rates.

Given the low acceptance rates, we could be tempted to submit one paper to multiple journals or conferences at once, without waiting until it is rejected in the first venue. However, *simultaneous submission* is strictly forbidden in the academic sphere, and it can result in a permanent publishing ban. We should also refrain from submitting our papers to so-called *predatory journals* that publish low-quality papers, fake the review process, spam researchers, or have hidden fees. Beall's List can act as one of the hints to decide.

10.3 Process Overview

This section summarizes the whole process from the point of finishing a paper draft to the point of having the paper published. Although we mention two typical procedures, one for journals and one for conferences, in reality many venues use something in between. We always need to consult the venue's website for instructions.

10.3.1 Journals

This is a typical process applied in journals, which often lasts from a few months to a year:

1. We select an appropriate journal and modify our draft to use the supplied template.
2. When we think the paper is ready for review, we submit it using the journal's online submission system.
3. The *editor* of the journal sends review invitations to researchers that could be interested in the paper. The selection is performed using personal knowledge, the submission system's database, paper references, or our suggestions.
4. Reviewers that accept the invitation read our article and write textual reviews. A typical number of reviewers is two to four, usually three. Review deadlines typically range from two weeks to two months.
5. The reviewers also supply suggestions to accept, reject or revise the paper. The term *major revision* is sometimes used if the reviewer will need to see the revised paper to decide, while *minor revision* means the reviewer is already decided the paper should be accepted after a revision.
6. The editor then makes a final decision based on the recommendations of the reviewers.
7. If the decision is to revise the paper, we make the suggested changes to the paper. We submit it along with a Response to Reviews document, which should contain answers to the reviewer's comments linked with the descriptions of the corresponding changes made in the paper.
8. The editor sends the revised version of the paper to the reviewers, as necessary. The process repeats from point 4 until the paper is accepted or rejected.
9. In the case of rejection, the process ends here. We should substantially improve it and re-submit it or submit it to another journal. If the paper is accepted, it is sent for "production" (copyediting and typesetting).
10. The accepted paper is put online by the publisher.
11. The paper is assigned to a specific issue of the journal. At this point, it is considered published.

10.3.2 Conferences

For conferences, the typical process is a bit different:

1. We find a conference with suitable deadlines, topic, and location.
2. We select a suitable track of the main conference (Research Papers, Tool Papers, ERA, etc.) or a co-located conference or workshop.
3. Our manuscript must be modified to respect the template and the page count specified by the conference track. We submit it using the submission system until a firm deadline.

4. Members of the *program committee* bid for the papers, expressing how much they want to review them by numeric scores.
5. Usually three reviewers are assigned to our paper using these scores.
6. Each reviewer submits a textual review and a recommendation on a given scale, such as reject, weak reject, weak accept, or accept.
7. The reviewers discuss until they come to a decision: either accept or reject the paper.
8. Based on the discussion, *program chairs* make the final decision.
9. Until the approximate deadline displayed on the website, we receive a *notification* containing the decision and the text of the reviews.
10. If the paper is rejected, the process ends here; we can improve it and submit elsewhere. In the case of acceptance, we submit a final "camera-ready" version until a deadline. This version is not checked by the reviewers again.
11. We register for the conference, travel to the given location, and present the paper.
12. The *conference proceedings* with all presented papers are published.

In recent years, some top-tier conferences have adopted a two-phase review process with revisions, similar to journals.

10.4 Reviewing

In academia, there is no single authority that reviews papers; instead, other researchers from our field (peers) do it. Therefore, we call it *peer review*.

The purpose of peer review is twofold. First, we decide whether the paper is ready for publication. This should, in theory, prevent incorrect or irrelevant papers from appearing in journals or conferences. Second, we try to increase the quality of the paper by providing comments.

Journals and conferences often list specific criteria upon which the reviewers should judge papers. As an example, we can mention the ICSE 2020 criteria: soundness, significance, novelty, verifiability, and presentation. In general, we can try to answer the following questions (loosely based on Zobel, 2014):

- Is the motivation, contribution, and research question described clearly?
- Is the contribution significant enough to be published?
- Is the method described clearly? Could we perform the study ourselves given such a description?
- Is the methodology valid?
- Do the results seem correct?
- Does the conclusion answer the research question?
- Is related work described in sufficient quantity and quality?
- Is the paper logically structured and well-written?

Some venues provide a specific structure that the text of the review should have, but generally we can expect a brief summary of the paper, a list of its strong points, description of its weak points, and suggestions for improvement.

Traditionally, the review process is *single-blind* (or single-anonymous), which means reviewers can see the authors' names, but not vice versa. In *double-blind* review, the authors have to anonymize their names in the paper and refer to their own works in a way that does not reveal their identity. The texts of the reviews are private in either case. *Open review*, which is still relatively rare, publishes the reviews alongside the papers, sometimes even with the reviewers' names. Open reviews for some conferences are available at OpenReview.net. In the PeerJ Computer Science journal, reviews are available thanks to the "Read the peer review reports" button at the beginning of the corresponding article's page. We recommend reading a few reviews to see how they typically look.

Regarding the ethical issues, we should not review papers for which we have a conflict of interest: advisorship, sharing the same institution, or a recent co-authorship with one of the authors. The papers that are currently under review are usually confidential, i.e., we should not disclose their content to others or publish the discussed ideas in any way.

10.5 Publication Access Models

In the past, publishers physically printed journals or conference proceedings and sold them to institutions and readers. Typesetting, printing, and distributing the physical copies incurred substantial costs. Institutions, such as universities, subscribed to journal bundles and paid for these costs. The publishers did not pay authors as the profit margins were not extremely large, and the authors were interested in reputation more than profit. To ensure profitability, the publishers required the authors to transfer their copyright to them or sign an exclusive copyright license, which means their rights to distribute papers were very limited, even the ones they wrote themselves.

With the advancement of the Internet, the publishers started to replace printed publications with *digital libraries*. The costs decreased dramatically, but the subscription prices were growing steadily. The publishers did not want to give up their *subscription-based* access model, based on *paywalls*. The readers need to access papers through a subscribed institution, otherwise they would need to buy individual articles for inappropriately high prices, of which nothing goes to the authors (and reviewers).

The *open access* principle tries to change this situation with multiple new models. *Green open access* means a paper is published as usual, behind a paywall, but the authors can put a free copy of their manuscript (so-called *pre-print* or *post-print*) on their website or selected repositories. Usually the publisher imposes a list of requirements, such as the version of the paper that must be used and phrases that have to be pasted into the paper. The authors should always read a specific license they are signing, but the Jisc

Open policy finder is useful for an overview. For all researchers, we definitely recommend creating your own website, however trivial, and uploading all papers compatible with green open access (nowadays almost all) to it. Additionally, we recommend uploading to arXiv, which makes the paper permanently accessible, provided the publisher allows this.

In *gold open access*, the publisher itself makes the papers accessible to everyone, without a paywall. Many publishers, however, ask an *article processing charge* (APC) for such a service, which is often an exorbitant sum.

10.6 Referencing Published Papers

After a paper is published, we can refer to it using a bibliographic citation. Its format depends on the type of the publication. For both journal and conference articles, the reference contains the authors' names, the title of the article, the year, and page numbers or article number. In addition to this, a journal article reference includes the journal's name and volume and/or issue, while a conference paper reference contains the name of the conference proceedings.

To globally identify a paper (or any document), publishers assign it a *Digital Object Identifier* (DOI). It is a unique permanent handle in the form `10.xxxx/...`. Visiting the DOI's corresponding URL, e.g., https://doi.org/10.1007/s10664-021-10083-5, redirects us to the official publisher's version of the paper. Including a DOI in bibliographic citations is recommended or required by many publishers.

A journal as a whole, i.e., not its specific issue or paper, can be identified by an *International Standard Serial Number* (ISSN). It is an 8-digit number assigned to periodical publications. A conference proceedings volume is assigned an *International Standard Book Number* (ISBN), which has 10 or 13 digits.

10.7 Data Deposition

Even after research findings are published, they should be subject to further scrutiny by other researchers to confirm their correctness. This is achieved through reproduction and replication of studies. According to ACM Artifact Review and Badging Version 1.1, *reproducibility* is the ability to obtain the same results by a different team with the same experimental setup, which includes the artifacts (programs and data) supplied by the original authors. On the other hand, *replicability* means the ability to obtain similar results using artifacts developed by the new team independently.

Publishing the artifacts, i.e., programs and data, used during research along with the paper is therefore becoming the norm. Uploading the data to a personal website or a shared drive and linking it in the paper's text is better than nothing. However, these links

often become invalid after some time. Thanks to specialized research *data repositories*, such as Zenodo, OSF, or Figshare, we can deposit data permanently and even assign it a DOI.

When depositing programs and data, we may be faced with the question of a license selection. To understand this issue, we have to explain the notion of copyright. *Copyright* is a legal protection of creative works such as text, images and software. Contrary to a popular belief, the purpose of a *license* is not to restrict otherwise unlimited rights. Instead, all new creative works are copyrighted by default. To allow others to reasonably use and reuse our works, we must explicitly declare a license. For software, we may use a website to choose an open source license. For text or images, Creative Commons offers a variety of licenses with different terms. If we are not the sole authors of the artifacts, for instance, if we transformed existing data, we need to take the original license into account during our decision.

Exercises

1. What are the publication types of these documents?
    a. A Large-Scale Study of Test Coverage Evolution
    b. Fragment-based spreadsheet debugging
    c. Insurers, spreadsheets and model risk
2. How could we speed up the traditional submission and publication process of journals or conferences without compromising review quality?
3. Why do you think simultaneous submission is prohibited?
4. Is the current peer review practice effective in preventing papers with factual errors from appearing? Why or why not?
5. What are the advantages and disadvantages of single- and double-blind review?
6. Why do you think the subscription-based model and the gold open access model with high APCs are still widely used when nowadays the price of hosting papers online is negligible?
7. Find a published replication study in your field. Did the replication confirm the original results?

11 Academic Practice

The final chapter of this book summarizes miscellaneous practical topics related to academia, namely presenting our findings, financing research, and academic career.

11.1 Presenting

There are many opportunities to present our research, including conferences, thesis defenses, and project summaries for various committees. Similar to writing a paper, where we empathized with our readers, we have to ask one important question before preparing a presentation: Who is my audience? Then we need to design our presentation on a level appropriate for the audience: explaining that Haskell is a functional language at a functional programming conference might seem condescending, while a grant committee including non-computer engineers would require a longer explanation of the context.

Two common forms of presentation are talks, usually accompanied by slides, and posters.

Posters often appear at conferences, where they are typically used either to present early-stage research or to complement a full-paper talk. An ideal poster should present main points, visible from a greater distance, and explain some details, but not as much as the whole paper. The presenter can provide further details when asked by other attendees. Perhaps the most important rule about posters is to prefer graphical content over text. For more ideas, we can refer to the tips for a good poster.

In the next subsections, we provide guidelines for *talks* with slides, mainly inspired by Zobel (2014) and the author's own experience.

11.1.1 Content

In a relatively short presentation, it is best to present only one main idea and its prerequisites, i.e., facts necessary for the understanding of this idea. A presentation should include at least motivation, possibly with an example, the designed approach (if any), a brief but clear overview of methods, main results, and conclusion. We can mention the most important related works directly motivating our research, not listing them extensively.

Generally, it is difficult for the audience to be focused during the whole presentation. The audience will quickly lose track if we include non-essential technical details, such as long listings of auxiliary algorithms or proofs of too many lemmas. For a similar reason, we should minimize the amount of content that the audience must remember from previous slides.

A common beginners' mistake is to allocate too much time for context, motivation, or some other specific part of the presentation. The individual parts should be balanced unless there is a specific reason not to do so.

11.1.2 Form

A general guideline is to prefer figures over text where possible, as they are generally easier to process by quick glances while the speaker talks. Not everyone has perfect sight, so we should use a large enough font, not trying to squeeze everything on one slide. Screenshots of tables from the paper tend to be particularly unreadable, so it is best to rewrite them using the presentation's style and simplify them if desirable.

Unless we deliver an hour-long keynote, starting with a Table of Contents slide is not necessary because it consumes valuable time from our presentation. Similarly, ending a presentation with an empty "Thank you. Questions?" wastes space and leaves the attendees without context. Instead, we can show a slide with a summary of the main points to spark a conversation.

11.1.3 Delivery

Presentations usually have a strictly allocated time that needs to be respected. As a safe estimate, we can plan about one slide per minute. In the case of shorter slides, the tempo can be increased up to two slides per minute. Ideally, we can rehearse the whole presentation and measure time. If we do not have this possibility, we can put less important stuff at the end of the presentation if it makes sense.

At the very least, we should imagine in our minds what we will talk about each slide. After conference talks and thesis defenses, questions tend to follow, so it is helpful to prepare for the most probable questions.

11.1.4 Tools

The whole look and feel of the presentation is shaped by the tool that we select to create it. For instance, if we create our presentation in a text editor using an HTML- or Markdown-based framework, we can be tempted to use more text and fewer images. As an opposite extreme, a vector graphical editor will tempt us to draw a lot.

The selection of the best tool thus depends on the specific situation and our goals. As a default choice, for a presentation that neither impresses nor disappoints, we recommend using standard slide-based tools, such as LibreOffice Impress or PowerPoint. They also come in handy for tight deadlines because we do not have to learn a new tool with unexpected glitches. Presentations heavy on math or containing LaTeX-produced diagrams benefit from the LaTeX's beamer package. To grab attendees' attention, we can try some of the alternative applications, e.g., Canva or Prezi if we have a time and desire to experiment. Finally, for interactive content embedding and custom scripting, frameworks such as reveal.js or Slidev are the best fit.

11.2 Financing Research

Research has its expenses, which include the staff salaries and students' stipends, material (hardware, software, books, etc.), travel to conferences, open access fees, and others. The primary sources of research funding are regular public funding of universities and research institutes, public grants, private donations and grants, and tuition fees. The next two sections discuss grant applications and research evaluation, which is often used to allocate public funding.

11.2.1 Grants

A *research grant* is a financial award, usually bound to a certain topic and time frame, given to a researcher or a group of researchers by a grant agency or other organization. To try receiving a grant, we need to choose an appropriate *call* and submit a *grant proposal*. Each grant call specifies exact requirements, but generally the grant proposal contains:

- a context, problem statement, and literature overview,
- a sketch of our proposed solutions,
- a budget listing the expected expenses,
- information about a *principal investigator* and other members of the research team,
- and the investigators' past achievements: publications and past grants.

After receiving a grant, we are usually expected to submit annual or final reports with achieved results. Most grants also require us to mention their number in the Acknowledgment section of each paper that was supported by them.

If we have a paper accepted at a high-quality conference but do not have sufficient funds to travel there, as a last resort we can sometimes apply for a corresponding ACM SIG fund, e.g., SIGSOFT CAPS for software engineering conferences.

11.2.2 Research Evaluation

Research funding is often allocated according to the researchers' achievements, either directly by dividing a fixed sum of money according to some quantitative criteria or indirectly by preferring high-performing researchers in grant applications. However, how to quantitatively measure the quality of research?

Perhaps the most obvious way to measure a researcher's output is to simply count the total number of publications. Of course, this favors many low-quality publications, so other metrics are necessary. The total number of citations of a researcher can give us some indication of quality, assuming high-quality works get more citations. A metric called *h-index* was devised to combine both publication and citation counts. A researcher having an h-index of h has h publications that have at least h citations each. Relying on citations is, in turn, susceptible to unethical citing by unrelated colleagues' papers.

A commonly used evaluation criterion is the number of the researcher's publications weighted by the quality of their publication venue. Some of the possible (but often flawed) proxies of the venue's quality are:

- its type: a journal or conference proceedings,
- whether it is indexed in databases such as Web of Science or Scopus,
- an *impact factor* for year Y of the journal (often calculated in the summer of $Y+1$), which is the total number of citations in year Y to the journal's papers from years $Y-1$ and $Y-2$ divided by the number of journal's papers published in $Y-1$ and $Y-2$ (Dong et al., 2005),
- and its *quartile* in the list of journals on a given topic indexed by a given citation database, ordered by an impact factor or another metric.

11.3 Academic Career

In this final section, we outline the practical aspects of pursuing PhD studies in computer science and possible further steps.

11.3.1 PhD Studies

The main goal of PhD studies is to learn how to become a researcher and, at the same time, do real research. According to the illustrated guide to a Ph.D., it is about getting to the boundary of human knowledge and pushing until it gives way.

The typical length of a PhD program in Europe is three to four years. Specifically, in Slovakia, it is officially considered study instead of employment for many purposes, e.g., students receive a tax-exempt monthly stipend instead of a salary, but from a practical standpoint, it is an equivalent of a full-time job.

As Feamster claims in his essay Do You Need a Ph.D.?, the main point is that people do not actually *need* a PhD. The journey is more important than the destination; intrinsic motivation plays a central role. The main reasons why someone should decide to do a PhD and the corresponding questions are:

- Discovery: Do I like to work on uncertain (ill-defined) problems?
- Expertise: Would I like to get a very deep understanding of one area and awareness about a wide range of other areas?
- Transferable skills: Would I like to strengthen my skills such as problem structuring, critical thinking, communication, teaching and mentoring, or independent work?
- Autonomy: Do I like to decide what to work on and how?

On the other hand, bad reasons to pursue PhD studies are:

- Financial status: Obviously, PhD is not a way to get rich quickly. (However, if money is the only problem, it should not discourage potential applicants.)
- Career status: Being called a "doctor" is cool but not a good main reason to do a PhD.
- To take more classes: There are usually almost no classes.
- As a fallback: It does not make sense to do a PhD only because someone is not ready for a real job.

Typical activities during PhD studies in computer science include reading research papers and books, discussing ideas with peers, implementing ideas and evaluating them, writing and submitting papers, traveling, presenting, teaching a few hours a week, supervising bachelor students, and writing a PhD thesis. A great activity to engage PhD students, especially in their beginnings, are reading seminars, where members of a research group read, present, and discuss a few relevant papers every week or two.

11.3.2 Further Steps

After finishing a PhD, it is possible to continue in industry or take the academic path. Possible academic career paths differ between countries and institutions. A *postdoc* is typically a temporary research-only position, while *assistant professors* tend to mix teaching and research. The two final steps are usually an *associate professor* and a *full professor*.

Exercises

1. Which of the mentioned tips for presenting will help you the most? Or did you already know all of them?

2. When deciding who gets a grant, a list of the investigators' past grants has a certain weight. This seems recursive: to get a grant, you need to have already received another grant. How do you think this is solved in practice? Do you suggest any modifications of such rules?
3. High-value grants are very competitive, and a fair selection of recipients difficult. Because some researchers have spent considerable time writing grant proposals without success, random grant allocation was even proposed (Barlösius & Philipps, 2022). Do you consider this a viable alternative? Why or why not?
4. What alternative ways of research evaluation not mentioned in this chapter exist?
5. Do you consider applying for PhD? What are your main reasons to apply and not to apply?

Assignments

Assignment 1: List of Papers for a Systematic Review

Imagine you are doing a systematic review (see Section 3.7). Leave out data analysis and synthesis – perform only the initial steps:

- define the topic and research question(s),
- specify the search strategy in detail,
- define inclusion and exclusion criteria,
- perform the search,
- and list the found relevant articles matching the criteria.

The narrower the topic, the less work you will potentially have during filtering of the papers; recall that a systematic review aims to ideally include *all* relevant works. However, there should be at least 20 relevant papers in the resulting list. If inclusion criteria specify a time range, it should span at least 5 years.

Your goal is to achieve as high recall and as high precision as possible with respect to all published research papers relevant to the given topic/question and matching the criteria. Since the set of all relevant published papers is unknown, the reviewer (teacher) can only guess it by assessing the quality of the research method and/or by performing random keyword searches.

You will be working in a team of two to four (ideally three) students, and you can use this fact to your advantage: for example, label a subset of papers by multiple persons, compute inter-rater reliability, and argue with this for the quality of your study. Alternatively, label all papers by two people and then resolve disagreements.

Semi-automated tools are allowed and encouraged. However, simply entering your research question into a chatbot and copying all results is definitely not enough. As the internal workings of the tool are opaque, you have to ensure the research method is valid, a large portion of relevant papers are included and irrelevant ones excluded.

The assignment should be submitted as a report in PDF format, containing a clear description of the method (including a diagram is helpful) and the list of relevant papers as bibliographic citations.

Assignment 2: Controlled Experiment Data Processing

Your task is to design and describe an imaginary controlled experiment. It will not be actually executed, so you can make up the data and state this in the report. Although unethical in practice, as an assignment this is a practical approach since executing a controlled experiment is often resource-intensive, and finding existing raw data from a controlled experiment without a paper already describing this experiment in detail is rare.

The report should describe at least:

- the research question, the null and alternative hypotheses,
- variables (including their scales),
- experimental design,
- details of the procedure,
- results (the made-up data, effect size, statistical testing),
- threats to validity,
- and conclusion.

The topic of the experiment should be related to computer science (interdisciplinarity is accepted). Quasi-experiments are allowed, but this has to be clearly stated in the report.

The experiment should be reported using a computational notebook such as Jupyter, marimo, or Observable Notebooks. A ZIP file should be submitted, containing:

- notebook source files (*.ipynb, *.py, etc.),
- data files, if any (e.g., *.csv),
- an *HTML (including images) or PDF export* of the notebook.

Assignment 3: Review of a Paper

Traditionally, a review contains a plain-text description of the paper's main point, strengths, weaknesses, and suggestions for improvement (see Section 10.4). The paper is judged using coarse-grained and vague criteria, such as novelty, validity, and presentation. This leads to very subjective reviews and a low level of agreement between multiple researchers reviewing the same paper.

To tackle this, a team of software engineering researchers led by Paul Ralph devised the ACM SIGSOFT Empirical Standards. Although the standards are primarily intended for software engineering, they encompass a wide range of research methods, which makes them usable for many other subfields of computer science. The Empirical Standards are based on checklists. First, the reviewer selects research methods that the paper uses. The tool then generates a list of specific attributes, such as "states formal hypotheses"

or "justifies sample size". The reviewer responds "yes" or "no" to each item; in the case of a negative answer, the system asks further questions. After submitting the form, the review result (acceptance or rejection) is generated. To learn more about the standards, see, for instance, a paper by Arshad et al. (2021).

In this assignment, you will select a computer science research paper from arXiv:

- at least 10 single-column or 8 two-column pages long,
- less than 12 months old,
- that probably has not yet been reviewed.

The last point can only be guessed, but generally this paper should not have an "accepted/published at" comment on arXiv, and searching the web for its title should not lead to a final published version. Then generate a checklist by selecting all methods that apply to the given paper. At least two methods must be checked; otherwise, please select another paper for this assignment.

Next, as the primary purpose of this assignment is not to accept or reject a paper but rather to assess your thinking, we will proceed a bit differently. Copy the generated attributes into a text document. In addition to choosing "yes" or "no" for each attribute, append a short note explaining your decision, such as the positions in the text where this criterion is fulfilled or a statement on why it is unfulfilled.

Grading will take into account whether the appropriate methods were selected in the generator and whether the decisions about attributes are correctly justified. You can ignore "desirable" and "extraordinary" attributes in the checklist. The maximum number of points for the assignment will then be limited to 12.

You should read your selected paper carefully, so you will be able to evaluate the attributes. You can use AI chatbots to learn, for example, what a given empirical standards attribute means in general, provided you verify the sources. However, it is forbidden to upload the paper to a chatbot and copy-and-paste its decisions on attributes.

The assignment should be submitted as a PDF file containing the paper's title, a link to it, the list of methods, and the review checklist as described above.

References

Abelson, H. (1986). *Lecture 1A: Overview and introduction to lisp [lecture transcript]*. MIT OpenCourseWare 6.001 Structure and Interpretation of Computer Programs. https://ocw.mit.edu/courses/6-001-structure-and-interpretation-of-computer-programs-spring-2005/resources/1a-overview-and-introduction-to-lisp/

Arshad, A., Ghaleb, T., & Ralph, P. (2021). Towards a more structured peer review process with empirical standards. *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, EASE '21*, 353–358. https://doi.org/10.1145/3463274.3463359

Arvanitou, E.-M., Ampatzoglou, A., Chatzigeorgiou, A., & Carver, J. C. (2021). Software engineering practices for scientific software development: A systematic mapping study. *Journal of Systems and Software*, *172*, 915–929. https://doi.org/10.1016/j.jss.2020.110848

Barlösius, E., & Philipps, A. (2022). Random grant allocation from the researchers' perspective: Introducing the distinction into legitimate and illegitimate problems in bourdieu's field theory. *Social Science Information*, *61*(1), 154–178. https://doi.org/10.1177/05390184221076627

Barua, A., Thomas, S. W., & Hassan, A. E. (2014). What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*, *19*(3), 619–654. https://doi.org/10.1007/s10664-012-9231-y

Begel, A., & Zimmermann, T. (2014). Analyze this! 145 questions for data scientists in software engineering. *Proceedings of the 36th International Conference on Software Engineering, ICSE 2014*, 12–23. https://doi.org/10.1145/2568225.2568233

Beller, M., Spruit, N., Spinellis, D., & Zaidman, A. (2018). On the dichotomy of debugging behavior among programmers. *Proceedings of the 40th International Conference on Software Engineering, ICSE '18*, 572–583. https://doi.org/10.1145/3180155.3180175

Blackburn, S. M. et al. (2006). The DaCapo benchmarks: Java benchmarking development and analysis. *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, 169–190. https://doi.org/10.1145/1167473.1167488

Booth, W. C., Colomb, G. G., Williams, J. M., Bizup, J., & FitzGerald, W. T. (2016). *The craft of research* (4th ed.). University of Chicago Press.

Burns, R. B. (2000). *Introduction to research methods* (4th ed.). SAGE Publications.

Carrera-Rivera, A., Ochoa, W., Larrinaga, F., & Lasa, G. (2022). How-to conduct a systematic literature review: A quick guide for computer science research. *MethodsX*, *9*, 101895. https://doi.org/10.1016/j.mex.2022.101895

Carvalho, L., Degiovanni, R., Cordy, M., Aguirre, N., Le Traon, Y., & Papadakis, M. (2024). SpecBCFuzz: Fuzzing LTL solvers with boundary conditions. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24*. https://doi.org/10.1145/3597503.3639087

Choudhuri, R., Liu, D., Steinmacher, I., Gerosa, M., & Sarma, A. (2024). How far are we? The triumphs and trials of generative AI in learning software engineering. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24*. https://doi.org/10.1145/3597503.3639201

Claes, M., Mäntylä, M. V., Kuutila, M., & Adams, B. (2018). Do programmers work at night or during the weekend? *Proceedings of the 40th International Conference on Software Engineering, ICSE '18*, 705–715. https://doi.org/10.1145/3180155.3180193

Creswell, J. W., & Creswell, J. D. (2018). *Research design: Qualitative, quantitative, and mixed methods approaches* (5th ed.). Sage.

Denning, P. J. (2005). Is computer science science? *Commun. ACM*, *48*(4), 27–31. https://doi.org/10.1145/1053291.1053309

Dong, P., Loh, M., & Mondry, A. (2005). The "impact factor" revisited. *Biomedical Digital Libraries*, *2*(1). https://doi.org/10.1186/1742-5581-2-7

Dubey, R. K., Thrash, T., Kapadia, M., Hoelscher, C., & Schinazi, V. R. (2021). Information theoretic model to simulate agent-signage interaction for wayfinding. *Cognitive Computation*, *13*(1), 189–206.

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting empirical methods for software engineering research. In F. Shull, J. Singer, & D. I. K. Sjøberg (Eds.), *Guide to advanced empirical software engineering* (pp. 285–311). Springer London. https://doi.org/10.1007/978-1-84800-044-5_11

Gray, J. (1992). *Benchmark handbook: For database and transaction processing systems*. Morgan Kaufmann Publishers Inc.

Habiba, U.-., Habib, M. K., Bogner, J., Fritzsch, J., & Wagner, S. (2024). How do ML practitioners perceive explainability? An interview study of practices and challenges. *Empirical Softw. Engg.*, *30*(1). https://doi.org/10.1007/s10664-024-10565-2

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, *38*(6), 1276–1304. https://doi.org/10.1109/TSE.2011.103

Hoda, R., Noble, J., & Marshall, S. (2013). Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering*, *39*(3), 422–444. https://doi.org/10.1109/TSE.2012.30

Huang, Y., Wang, J., Liu, Z., Wang, Y., Wang, S., Chen, C., Hu, Y., & Wang, Q. (2024). CrashTranslator: Automatically reproducing mobile application crashes directly from stack trace. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24*. https://doi.org/10.1145/3597503.3623298

Huijgens, H., Rastogi, A., Mulders, E., Gousios, G., & Deursen, A. van. (2020). Questions for data scientists in software engineering: A replication. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*, 568–579. https:

//doi.org/10.1145/3368089.3409717

Huppler, K. (2009). The art of building a good benchmark. In R. Nambiar & M. Poess (Eds.), *Performance evaluation and benchmarking* (pp. 18–30). Springer Berlin Heidelberg.

Inal, Y., Clemmensen, T., Rajanen, D., Iivari, N., Rizvanoglu, K., & Sivaji, A. (2020). Positive developments but challenges still ahead: A survey study on UX professionals' work practices. *J. Usability Studies*, *15*(4), 210–246.

Inayat, I., Salim, S. S., Marczak, S., Daneva, M., & Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*, *51*, 915–929. https://doi.org/10.1016/j.chb.2014.10.046

Jedlitschka, A., & Pfahl, D. (2005). Reporting guidelines for controlled experiments in software engineering. *2005 International Symposium on Empirical Software Engineering, 2005.*, 1–10. https://doi.org/10.1109/ISESE.2005.1541818

Kabir, S., Udo-Imeh, D. N., Kou, B., & Zhang, T. (2024). Is Stack Overflow obsolete? An empirical study of the characteristics of ChatGPT answers to Stack Overflow questions. *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, CHI '24*. https://doi.org/10.1145/3613904.3642596

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining GitHub. *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, 92–101. https://doi.org/10.1145/2597073.2597074

Kampenes, V. B., Dybå, T., Hannay, J. E., & Sjøberg, D. I. K. (2007). A systematic review of effect size in software engineering experiments. *Information and Software Technology*, *49*(11), 1073–1086. https://doi.org/10.1016/j.infsof.2007.02.015

Kazemi, M. et al. (2025). BIG-bench extra hard. *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 26473–26501. https://doi.org/10.18653/v1/2025.acl-long.1285

Keshav, S. (2007). How to read a paper. *SIGCOMM Comput. Commun. Rev.*, *37*(3), 83–84. https://doi.org/10.1145/1273445.1273458

Kitchenham, B. A., Dyba, T., & Jorgensen, M. (2004). Evidence-based software engineering. *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference On*, 273–281. https://doi.org/10.1109/ICSE.2004.1317449

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (Technical Report EBSE-2007-01). Keele University; Durham University Joint Report. https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf

Kounev, S., Lange, K.-D., & Kistowski, J. von. (2025). *Systems benchmarking: For scientists and engineers* (2nd ed.). Springer. https://doi.org/10.1007/978-3-031-85634-1

Krause, A., Kaur, H., Klemmer, J. H., Wiese, O., & Fahl, S. (2025). "That's my perspective from 30 years of doing this": An interview study on practices, experiences, and challenges of updating cryptographic code. *34th USENIX Security Symposium*,

2907–2926.

Lamport, L. (2012). How to write a 21st century proof. *Journal of Fixed Point Theory and Applications*, *11*(1), 43–63.

Lawrance, J., Bogart, C., Burnett, M., Bellamy, R., Rector, K., & Fleming, S. D. (2013). How programmers debug, revisited: An information foraging theory perspective. *IEEE Transactions on Software Engineering*, *39*(2), 197–215. https://doi.org/10.1109/TSE.2010.111

Lubars, M. D. (1989). The IDeA design environment. *Proceedings of the 11th International Conference on Software Engineering, ICSE '89*, 23–22. https://doi.org/10.1145/74587.74590

Miao, X., Wu, Y., Chen, L., Gao, Y., & Yin, J. (2023). An experimental survey of missing data imputation algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *35*(7), 6630–6650. https://doi.org/10.1109/TKDE.2022.3186498

Munaiah, N., Kroh, S., Cabrey, C., & Nagappan, M. (2017). Curating GitHub for engineered software projects. *Empirical Software Engineering*, *22*(6), 3219–3253. https://doi.org/10.1007/s10664-017-9512-6

OECD. (2015). *Frascati manual 2015: Guidelines for collecting and reporting data on research and experimental development* (p. 398). OECD Publishing. https://doi.org/10.1787/9789264239012-en

Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST '23*. https://doi.org/10.1145/3586183.3606763

Roman, G.-C., & Cox, K. C. (1989). Declarative visualization in the shared dataspace paradigm. *Proceedings of the 11th International Conference on Software Engineering, ICSE '89*, 34–43. https://doi.org/10.1145/74587.74591

Rothlisberger, D., Harry, M., Binder, W., Moret, P., Ansaloni, D., Villazon, A., & Nierstrasz, O. (2012). Exploiting dynamic information in IDEs improves speed and correctness of software maintenance tasks. *IEEE Transactions on Software Engineering*, *38*(3), 579–591. https://doi.org/10.1109/TSE.2011.42

Saeki, M., Horai, H., & Enomoto, H. (1989). Software development process from natural language specification. *Proceedings of the 11th International Conference on Software Engineering, ICSE '89*, 64–73. https://doi.org/10.1145/74587.74594

Saunders, B., Sim, J., Kingstone, T., Baker, S., Waterfield, J., Bartlam, B., Burroughs, H., & Jinks, C. (2018). Saturation in qualitative research: Exploring its conceptualization and operationalization. *Quality & Quantity*, *52*(4), 1893–1907. https://doi.org/10.1007/s11135-017-0574-8

Shahin, M., Liang, P., & Babar, M. A. (2014). A systematic review of software architecture visualization techniques. *Journal of Systems and Software*, *94*(Supplement C), 161–185. https://doi.org/10.1016/j.jss.2014.03.071

Shreeve, B., Gralha, C., Rashid, A., Araújo, J., & Goulão, M. (2023). Making sense of the unknown: How managers make cyber security decisions. *ACM Trans. Softw. Eng. Methodol.*, *32*(4). https://doi.org/10.1145/3548682

Steimann, F. (2018). Fatal abstraction. *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2018*, 125–130. https://doi.org/10.1145/3276954.3276966

Stol, K.-J., & Fitzgerald, B. (2018). The ABC of software engineering research. *ACM Trans. Softw. Eng. Methodol.*, *27*(3). https://doi.org/10.1145/3241743

Tabassum, S., Pereira, F. S. F., Fernandes, S., & Gama, J. (2018). Social network analysis: An overview. *WIREs Data Mining and Knowledge Discovery*, *8*(5), e1256. https://doi.org/https://doi.org/10.1002/widm.1256

Vidoni, M. (2022). A systematic process for mining software repositories: Results from a systematic literature review. *Inf. Softw. Technol.*, *144*(C). https://doi.org/10.1016/j.infsof.2021.106791

Wobbrock, J. O., & Kientz, J. A. (2016). Research contributions in human-computer interaction. *Interactions*, *23*(3), 38–44. https://doi.org/10.1145/2907069

Wohlin, C., & Aurum, A. (2015). Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Softw. Engg.*, *20*(6), 1427–1455. https://doi.org/10.1007/s10664-014-9319-7

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2024). Systematic literature studies. In *Experimentation in software engineering* (2nd ed., pp. 51–63). Springer. https://doi.org/10.1007/978-3-662-69306-3_4

Yang, D., Martins, P., Saini, V., & Lopes, C. (2017). Stack Overflow in GitHub: Any snippets there? *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 280–290. https://doi.org/10.1109/MSR.2017.13

Yang, X., Lo, D., Xia, X., Zhang, Y., & Sun, J. (2015). Deep learning for just-in-time defect prediction. *2015 IEEE International Conference on Software Quality, Reliability and Security*, 17–26. https://doi.org/10.1109/QRS.2015.14

Zeller, A., & Lütkehaus, D. (1996). DDD—a free graphical front-end for UNIX debuggers. *SIGPLAN Not.*, *31*(1), 22–27. https://doi.org/10.1145/249094.249108

Zobel, J. (2014). *Writing for computer science*. Springer. https://doi.org/10.1007/978-1-4471-6639-9