



Empirical Software Engineering Research

Report Writing

Norman Peitek, Annabelle Bergum, Lina Lampel, Sven Apel

Learning Goals

- How to report on an (excellent) experiment
- Typical structural mistakes to avoid in a report
- Some basic tips on writing
- Some basic tips on data visualization

Report Structure

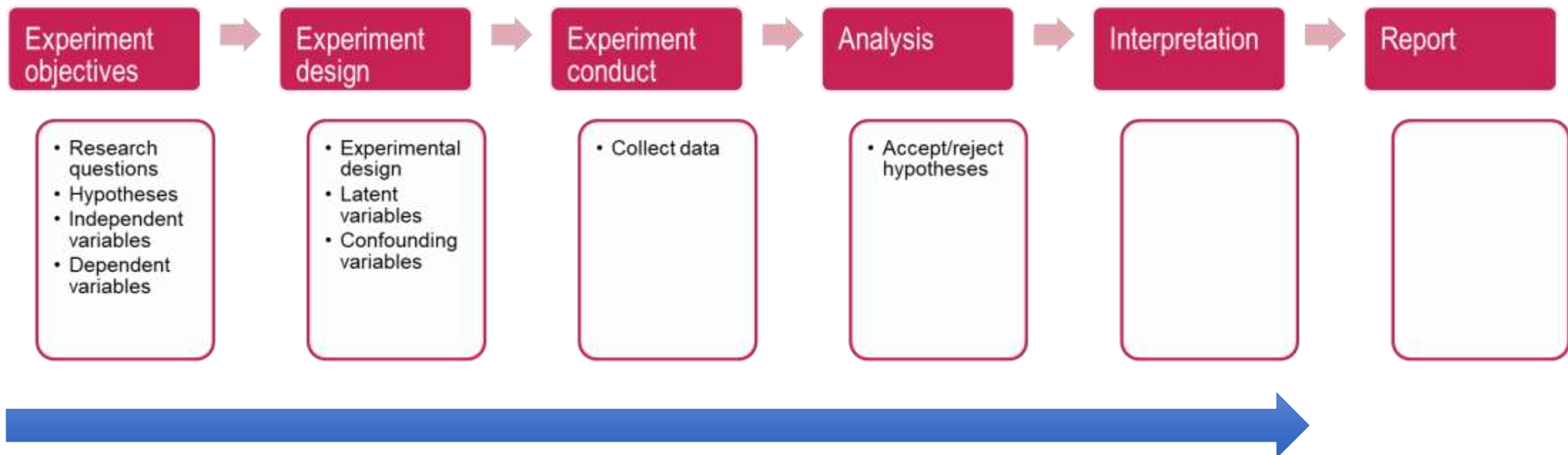


Overall Structure of a Report

- A reporting on an experiment can be a thesis, scientific paper, business case, ...
- It is critically important to precisely document experiments
- There are sensible conventions regarding the structure
 - Unless you have a good reason, it may hurt your thesis/paper when you break the convention

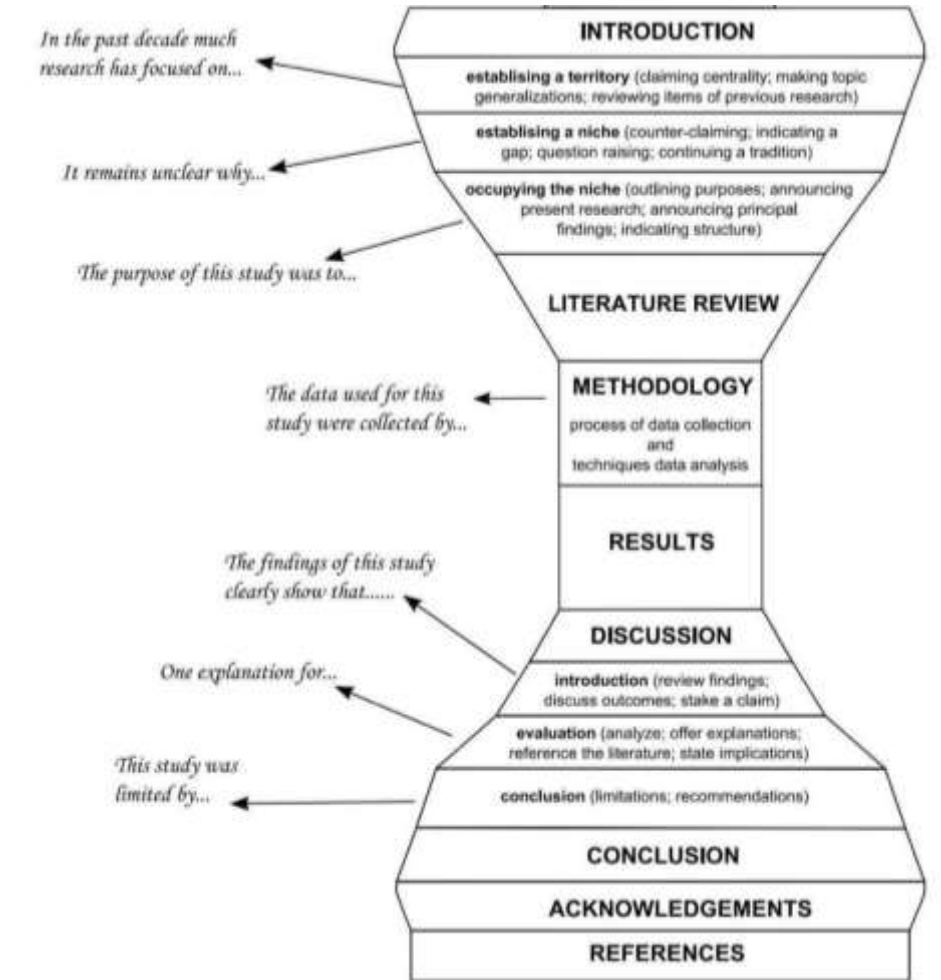
Overall Structure of a Report

- Generally, papers follow the structure of the experimental process



Overall Style

- Broad beginning
 - Software engineering → your topic
- Be specific in the main sections of the report
- Widen your horizon at the end again
 - Topic → software engineering



- Everything following is just guidelines!
 - Deviate from suggested structure if there are justifiable reasons
- Publication vendors in academia often have page limits, so some tradeoff decisions between accurate details and length must be made
 - If content does not fit in the paper, publish a replication package/website/extension with more information
 - For a thesis, sometimes moving information to an appendix is useful
- Upcoming screenshots are from: "What Drives the Reading Order of Programmers?"
 - <https://www.se.cs.uni-saarland.de/publications/docs/PSA19.pdf>
 - This paper (like any other) is not perfect and just serves as an example

- Establish importance of your topic with context and motivation for your experiment
 - If necessary, brief background information
 - Guide the reader to understand the problem
 - Short introduction to relevant knowledge/literature in this area
- Describe gaps in existing research
- Provide high-level information about your experiment
- Summarize the results of your experiment

Why should
someone read your
report?

Introduction

1 INTRODUCTION

In the past decades, much research has focused on how programmers comprehend source code, which is a central activity in software development [26, 52]. The underlying cognitive process, *program comprehension*, is a prerequisite for all subsequent programmer activities, such as testing, debugging, and maintenance. Past research theorized on two main strategies of how programmers comprehend software. *Bottom-up comprehension* is used when programmers lack domain knowledge, experience, or context to efficiently understand source code [39]. Instead, they have to understand individual source code lines and statements and integrate their semantic meaning to eventually build an overarching understanding (i.e., chunking [46]). *Top-down comprehension* is used when programmers take advantage of previous experience or domain knowledge for an efficient hypothesis-driven comprehension process [51], for example, guided by variable identifiers [10].

Although there is some evidence for the validity of these existing comprehension models, there are still knowledge gaps, such as when and how programmers are able to apply top-down comprehension. Program comprehension is an internal cognitive process and as such inherently difficult to measure [47]. Conventional methods, such as think-aloud protocols or measuring task efficiency, cannot provide deep insights into the underlying cognitive processes of program comprehension.

One important aspect of program comprehension is observing the way programmers *read* source code. Eye tracking has proved useful to observe programmers reading source code and answer such fundamental research questions on program comprehension (e.g., [13, 44, 53]). For example, Sharif and Maletic replicated a conventional study with eye tracking and found that naming style affects program comprehension in that programmers are able to read `under_score` style faster than `camelCase` style [8, 44].

Previous research suggested that the *linearity of the reading order* could be an indicator of how efficient programmers comprehend source code [13]. Busjahn et al.'s seminal study described several eye-gaze measures to gauge linearity of reading order. They showed that programmers read source code less linear than natural text and also that expert programmers read source code less linearly than novices [13]. This study indicates that comprehending source code is a skill that needs to be developed and honed with experience. A replication of Busjahn et al.'s study by Peachock et al. supports

Why is this paper relevant to software engineering?

Research gaps

Motivate methodology

Prior research

the adequacy of the developed eye-gaze measures and partially corroborated Busjahn's study results with student participants [38].

In this paper, we further dig into the role of the linearity of reading order for program comprehension. Specifically, we aim at understanding how programmers' comprehension strategy and linearity of source code itself affect programmers' reading behavior. Understanding all factors that influence programmers' linearity of reading order is critical to more accurately measure program comprehension with eye tracking. To this end, we conducted a non-exact replication of the studies by Busjahn et al. and Peachock et al. with novice and intermediate programmers. Our study differs in the following details: First, based on the two studies, we further refined the materials with a more systematically varied source code linearity. Second, we investigated in addition the interaction between comprehension strategy (i.e., top-down comprehension or bottom-up comprehension) with the linearity of reading order.

In short, we make the following contributions:

- We report on a non-exact replication of Busjahn et al.'s eye-tracking study on linearity of reading order.
- We provide further evidence that more experienced programmers read source code less linearly than novices.
- We present data that indicate that top-down and bottom-up comprehension affect linearity of reading order.
- We propose a method to systematize source code linearity and demonstrate that differences between source code snippets can substantially influence programmers' reading order.

Summarize paper

Contributions
(academic practice
in SE)

- For longer paper (in particular theses), outline the rest of the paper at the end of the introduction
- „In Chapter 2, we will In Chapter 3, we will..“
 - Especially useful if you deviate from common structure

1.3 Outline

Background In Chapter 2, we present the state of the art on models of program-comprehension strategies. We introduce which measurement methods researchers are using to observe program comprehension. We also provide an overview of how neuroimaging, eye tracking, and conventional measurements methods provide different perspectives on programmers' cognitive processes. In addition, we provide a close look at eye tracking and fMRI as measurement methods.

Methods and Results This dissertation's work is split into three parts (cf. Figure 1.1). First, Chapter 3 dives into our developed experiment framework, which shows how to conduct multi-modal fMRI experiments. We also present *CODERSMUSE*, a tool for multi-modal data exploration specific to the needs of software-engineering research. Next, Chapter 4 investigates the neuro-cognitive perspective of program comprehension in more detail. We present multiple studies on aspects of top-down comprehension as well as programmer expertise. Last, Chapter 5 shows three practical applications of our framework: An objective view on the relationship between code complexity metrics and programmers' cognition, a re-analysis investigating various aggregation levels of human responses, and an alternative fMRI analysis based on participant-specific anatomies.

- Introduce and explain all **relevant** special knowledge
 - Do not explain programming or software engineering
 - But, explain relevant specifics, such as program comprehension or continuous integration
- Establishes minimum knowledge for the readers to understand the rest of the paper
- Tip: Collect and *write down* relevant information during your literature review
 - But, do not share every paper with all details you have read
 - You must select before writing the background section

2 ORIGINAL STUDY AND REPLICATION

In this section, we briefly summarize the original study by Busjahn et al. as well as the replication study by Peachock et al.

2.1 Original Study (Busjahn et al.)

Busjahn et al. conducted a novel study on programmers' linearity of reading order [13]. They compared the linearity of reading order of novice and expert programmers as well as the novices' linearity of reading order for natural text and for source code. They observed the eye movements of 14 students while reading source code as well as natural text in their weekly Java beginners course. The natural text were short English passages of four to five lines. In addition, they asked 6 professional programmers to comprehend the source code and observed their eye movements. Due to the novelty of this research question, they also described appropriate eye-gaze measures to quantify the *linearity of reading order*.

Experiment Design. Ultimately, Busjahn et al. ran 17 trials of novices reading natural text, 101 trials of novices reading source code, and 21 trials of experts reading source code. As the novices were still learning programming, their snippets were simpler than the snippets for the expert participants. Only two snippets had to be comprehended by both participant groups. For all snippets, participants were randomly asked one of three possible tasks: write a summary of the source code, compute the output, or answer a multiple-choice question. To observe eye movements Busjahn et al. used a SMI RED-m remote eye-tracker with a sample rate of 120 Hz

Participants. 7 of the 14 novices were females. They were between 19 and 33 years old, had, at most, little programming experience, and all had, at least, a medium English proficiency (while German being the native language). The experts were all professional programmers with, at least, 5 years of programming experience, and were between 26 and 49 years old. One of the 6 experts was female.

Variables. The study of Busjahn et al. had two independent variables: programmer experience (novice or expert, between-subject) and, for novices, whether the presented stimuli were source code or natural text (within-subject). Busjahn et al. analyzed the data in two steps: First, they contrasted how novices read source code versus natural text (within-subject). Second, they contrasted linearity of reading order between experts and novices (between-subject).

Dependent Variables. To quantify the participants' linearity of reading order, Busjahn et al. describe a set of six eye-gaze measures, which we summarize in Table 1 and which we will also use for our data analysis.¹ In essence, Busjahn et al.'s eye-gaze measures abstract a fixation sequence of (x,y) coordinates on the screen to higher-level concepts, such as *regressions*. Regressions are a sign that a participant had to revisit a previous part, which could be due to an insufficient understanding or following a snippet's execution (e.g., loop structures).

Needleman-Wunsch Algorithm. In addition to the six fixation-based eye-gaze measures, Busjahn et al. analyzed the order in which each source code line was fixated and contrasted it with (a) the "story order" and (b) the execution order of a source code. The story order of a source code snippet is the sequence of each line from top to bottom, similar to natural text (e.g., 1, 2, 3, 4). The execution order of a source code snippet is the sequence of lines in which the code is executed, which may differ significantly from the story order (e.g., 3, 4, 2, 1, 2, 4).

The presented source code snippets contained up to 30 lines of source code. To effectively compare long sequences of line numbers, Busjahn et al. relied on the Needleman-Wunsch (N-W) algorithm, which was originally designed for molecular comparisons of proteins [34] and later applied for use in eye-tracking research by Cristino et al. [16]. The N-W algorithm computes the similarity between two sequences. In this study, it can be interpreted as how similar an observed linearity of reading order is to a line-by-line reading order or a computer's execution order of the source code. Furthermore, as programmers often cannot comprehend a piece of source code in a single read, Busjahn et al. added a dynamic version of the N-W algorithm that tolerates multiple reads. In our data analysis, we will also use both versions of the N-W algorithm to assess the linearity of reading order of our participants.

Results. Busjahn et al. reported two main findings: First, novice programmers read source code less linearly than natural text. Second, expert programmers read source code less linearly than novice programmers.

Objectives

- What are the goals of the experiment?
- Where do these goals come from?
- What would be the impact of answering them?
- Typically expressed in forms of research questions and sometimes hypotheses
 - Research questions if little prior knowledge is known
 - Provide evidence for the relevance of your research question

Ideally, you support your objectives with existing literature

- Was this research called for in prior publications?
- Show arguments for different reasonings for your research questions/hypotheses
- If applicable, predict findings based on existing knowledge

3 EXPERIMENT DESIGN

The overarching goal of our study is to gain a deeper understanding of how source code, programmer experience, and comprehension strategy affect linearity of reading order. We provide a replication package, which includes all stimuli, acquired data, and analysis scripts.² Specifically, we pose the following research questions:

RQ1: Can we resolve the contradicting results of Busjahn et al. and Peachock et al. regarding whether more experienced programmers read source code less linear than novice programmers?

RQ2: Does the comprehension strategy, that is, bottom-up and top-down comprehension, affect linearity of reading order?

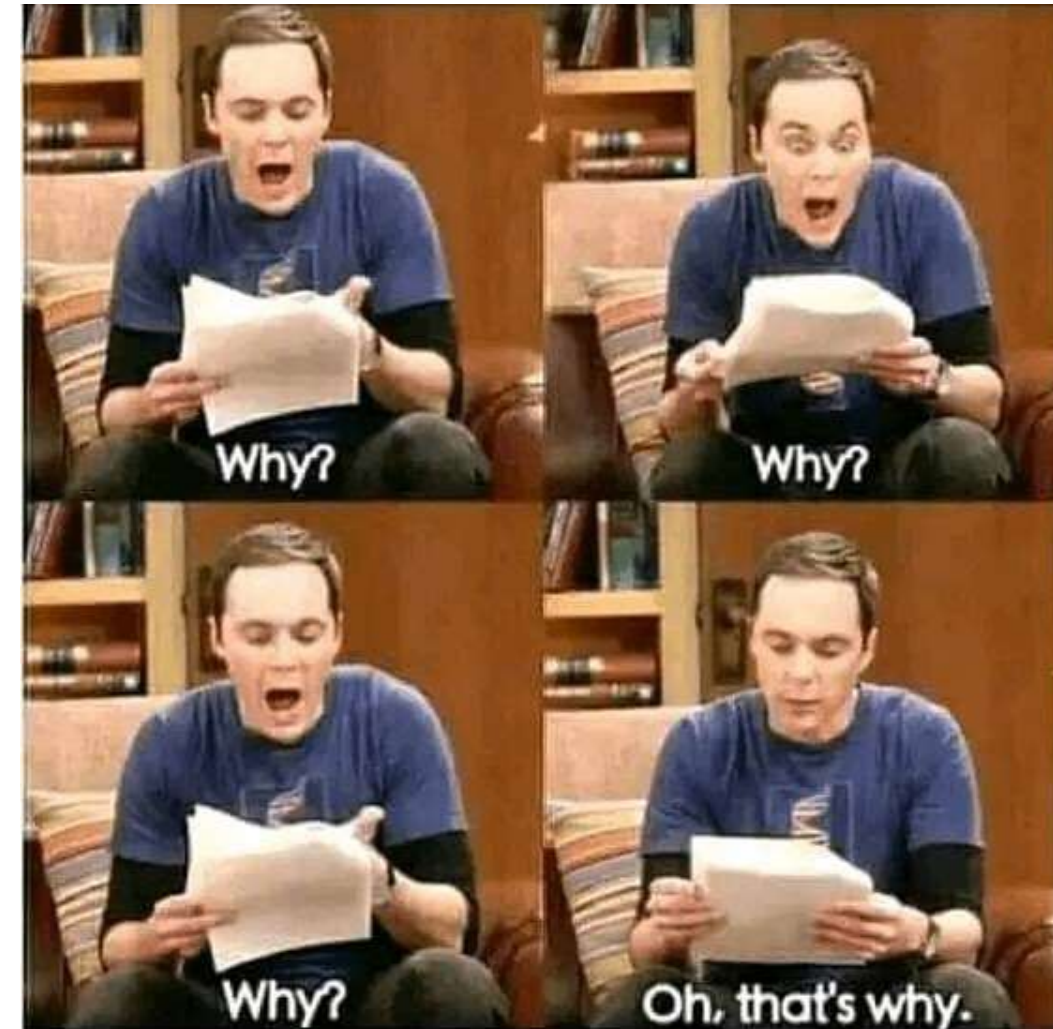
RQ3: Does the linearity of source code affect programmers' linearity of reading order?

To evaluate RQ1, we conducted a non-exact replication with novice programmers as well as more experienced programmers that can be classified as *intermediate* programmers according to Dreyfus' taxonomy of skill acquisition [18, 32]. Since we presented the same source code snippets to both groups, we can reduce the risk of a confounding factor arising from within the source code snippets.

- In the example paper, objectives is the first part of the experiment design section

Experiment Design: Material

- Describe all chosen materials
 - For example, source code snippets, tools, questionnaires, ...
 - Explain *why* it is a fitting choice for your experiment
 - If applicable, where did you take/adapt them from?



- How did you invite the participant/select the sample?
- Based on what criteria?
 - Motivate why there are the right sample for your research question
 - If you study an algorithm, justify the configuration/parameters
 - For example, JavaScript experts may not be a good sample to measure how difficult it is for beginners to learn TypeScript
- Describe the characteristics of your sample
 - Typically: age, gender, educational status, programming experience, ...
 - Include everything that could potentially be a confounding factor

- Describe what experiment design was used (use established terms!)
- Motivate why it is the correct choice
 - Sometimes, explain why the at a first glance an obvious choice is not ideal

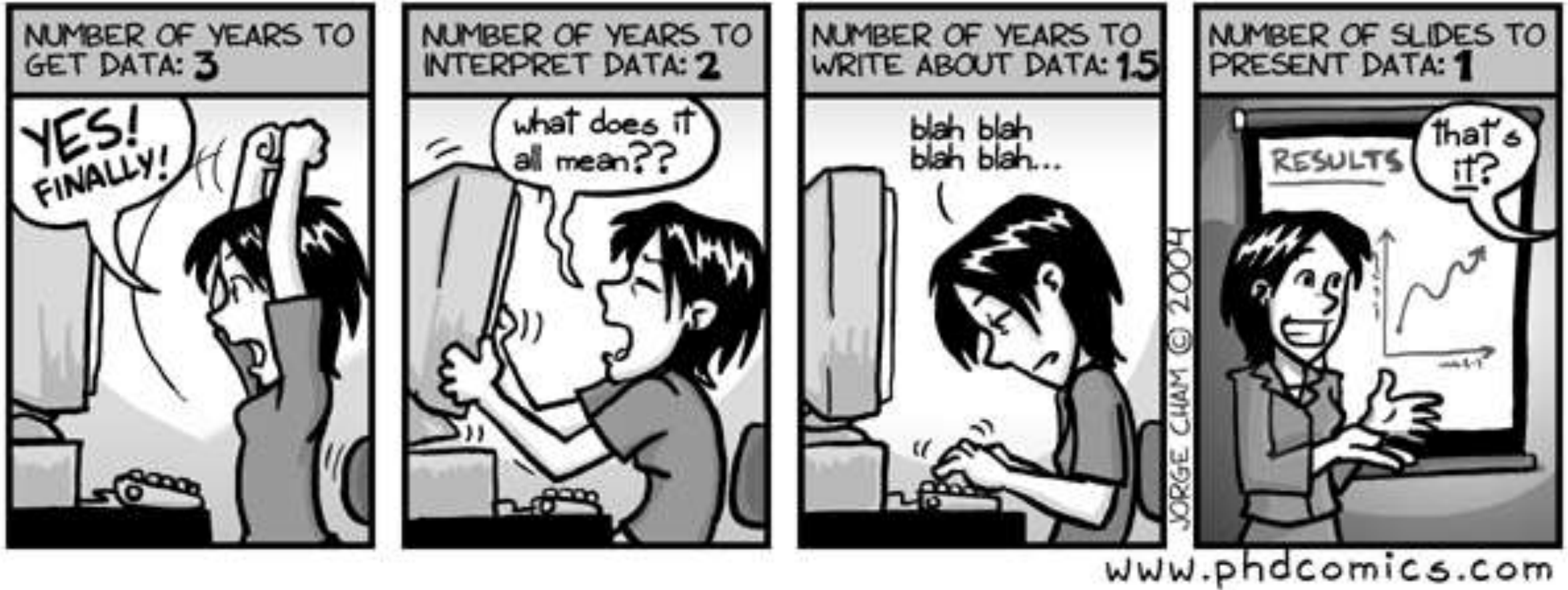
- How was the experiment conducted?
 - How did the participants learn what to do (written versus verbal explanation)?
 - Was there a training/warm-up? If so, why?
 - When did you show questionnaires?
 - Answers may be dependent on the placement in the experiment
 - For example, self-rating of skill could be biased if it's shown after an experiment that went (not) very well

- If there were deviations, state them clearly
 - Something can always go wrong
 - If the results are not too biased/problematic, they still might be interesting
 - Some researchers also report that there were no deviations
- For example, in the required reading 3 source-code snippets contained unintended syntax errors
 - This may have influenced the cognitive process of program comprehension we intended to observe
 - But, it (by accident) led to an interesting finding in itself

- Describe (pre)processing of the data
 - If you follow a standard or a prior paper, state and reference it
- If applicable, describe in detail and justify data (outlier) removal
 - Ideally, you set these criteria before conducting the study and based on the literature

- Describe the data with descriptive statistics
 - Provide a decent amount of information without being overwhelming (typically no raw data!)
 - Descriptive statistics: Measures of central tendency, standard deviations, ...
- Use digestible visualizations
 - Visualizations: Scatterplots, box plots, violin plots, ...
 - Generally, stick to common metrics and visualizations
 - One good visualization can make a report stand out

DATA: BY THE NUMBERS



- Describe the inference statistics and their results
 - For example, statistical significance test such as t-test
 - Justify your choice of inference methods
 - If necessary, include assumption tests for the significance tests
 - For example, parametric tests assume normal distribution of the data
 - Don't assume perfect statistical knowledge from the reader
 - Follow reporting guidelines for statistical tests on which values to include
 - Example: Coffee drinkers spent more time awake ($M = 17.8$, $SD = 1.4$) than the population norm, **$t(28) = 2.6$, $p = 0.005$** .
 - If possible, include effect sizes
- Answer your research question/hypotheses
 - Do **not** mix in your interpretation of the results!

- What do the results mean for your research question and the larger context?
 - Assess your own interpretation and compare it to alternative explanations
- Revisit the motivation for your study (gaps in research)
 - Remind the reader of your study design and setup
 - Summarize the results of your study
 - Tie in results from literature (how do your results confirm/contradict prior results)?
- Implications of the results for the field
 - Widen in scope and close the loop to the context of the introduction
 - Generalize your findings as far as possible (but no further)

- Limitations of your study and possible future work
 - Typically, the limitations provide a basis for future work
 - Sometimes part of the conclusion or its own section
 - Suggestions for future work are very valuable
- Lessons learnt
 - Share knowledge you gained even if it is outside of the posed research question
- Action plan
 - Sometimes useful for reports in the industry

- How is the validity threatened for your experiment?
- What measures did you take to mitigate potential threats?
 - For example, randomization of task order to prevent learning effects
- Many publication vendors require this section nowadays
 - Usually, at least internal and external validity is discussed
 - Sometimes construct validity and statistical conclusion validity are also relevant (still less common in SE)

- What have other researchers done in this research area?
- What are commonalities and differences to your study?
- Often some repetition from the discussion section (since you should tie in these to help integrate your results)
- Make sure to synthesize results and not just present individual papers
- Depending on the paper this section is not always necessary

Conclusion

- Typically, short summary of your paper
- Describe the effect of your study for the field

- Acknowledgments
 - Thank study participants and everyone who contributed to the study/paper
- References
 - Use automated system (bibtex in latex, Microsoft Word's integrated reference manager)
- Appendix
 - Typically, less important information, but should be included for completeness
 - For example, all data from a questionnaire, full results from an analysis if only a subset were shown in the results section, all stimuli, ...
 - Nowadays, often an external replication package (ideally permanently archived)

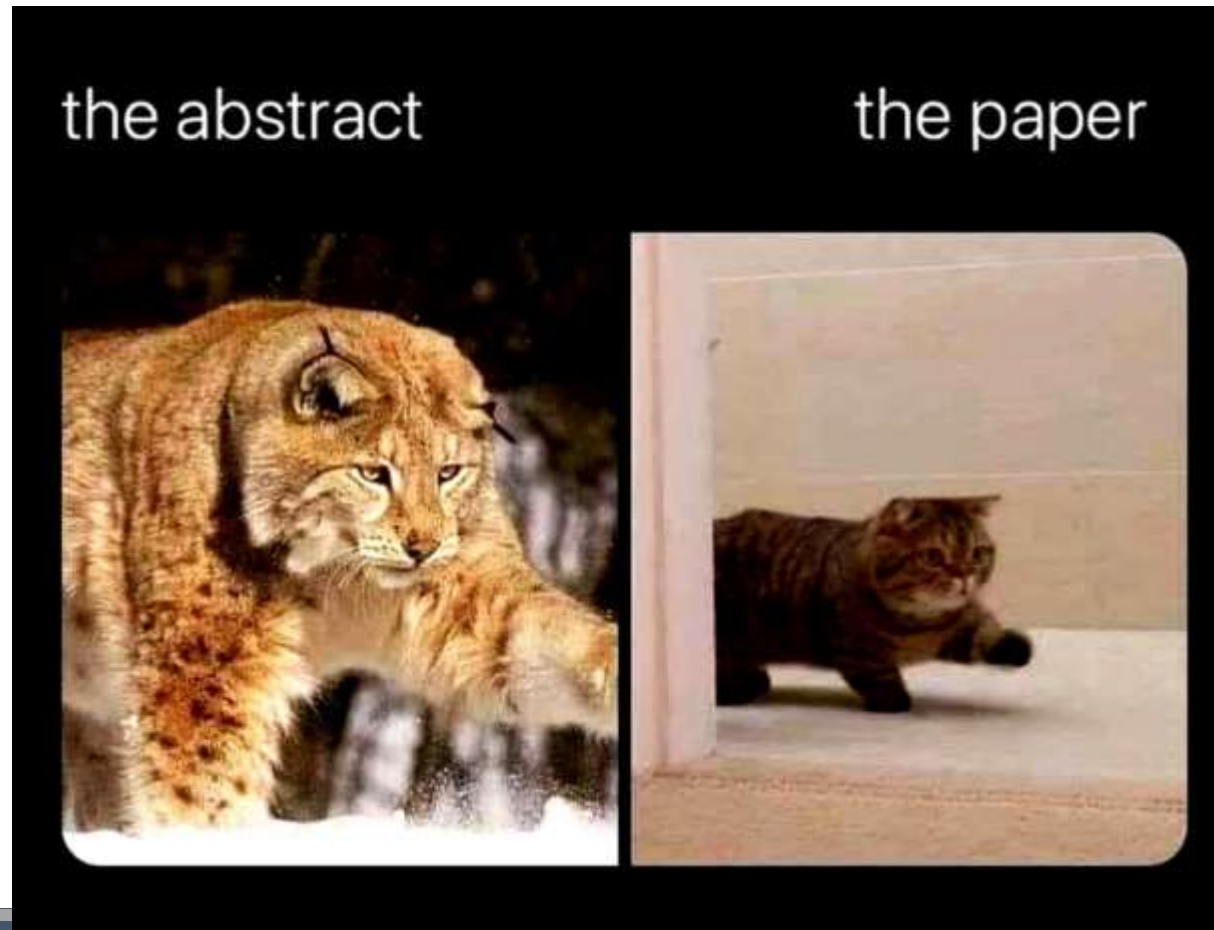
- Almost always required
 - Even for 2-page papers
- Typically, short summary of your paper, including context, motivation, results, implications
 - Should be representative of your paper
- „Executive summary“ in industry

ABSTRACT

Most modern software programs cannot be understood in their entirety by a single programmer. Instead, programmers must rely on a set of cognitive processes that aid in seeking, filtering, and shaping relevant information for a given programming task. Several theories have been proposed to explain these processes, such as “beacons,” for locating relevant code, and “plans,” for encoding cognitive models. However, these theories are decades old and lack validation with modern cognitive-neuroscience methods. In this paper, we report on a study using functional magnetic resonance imaging (fMRI) with 11 participants who performed program comprehension tasks. We manipulated experimental conditions related to beacons and layout to isolate specific cognitive processes related to bottom-up comprehension and comprehension based on semantic cues. We found evidence of semantic chunking during bottom-up comprehension and lower activation of brain areas during comprehension based on semantic cues, confirming that beacons ease comprehension.

Abstract

- The abstract should be representative of your paper



- *Structured* abstracts provide a skeleton similar to the structure of the paper
 - Guides readers' attention
 - Aids decision making whether to read a paper
- Some publication vendors/sciences require structured abstracts
 - Not as common in SE yet, but slowly increasing

ABSTRACT

Background: The way how programmers comprehend source code depends on several factors, including the source code itself and the programmer. Recent studies showed that novice programmers tend to read source code more like natural language text, whereas experts tend to follow the program execution flow. But, it is unknown how the *linearity of source code* and the comprehension strategy influence programmers' *linearity of reading order*.

Objective: We replicate two previous studies with the aim of additionally providing empirical evidence on the influencing effects of linearity of source code and programmers' comprehension strategy on linearity of reading order.

Methods: To understand the effects of linearity of source code on reading order, we conducted a non-exact replication of studies by Busjahn et al. and Peachock et al., which compared the reading order of novice and expert programmers. Like the original studies, we used an eye-tracker to record the eye movements of participants (12 novice and 19 intermediate programmers).

Results: In line with Busjahn et al. (but different from Peachock et al.), we found that experience modulates the reading behavior of participants. However, the linearity of source code has an even stronger effect on reading order than experience, whereas the comprehension strategy has a minor effect.

Implications: Our results demonstrate that studies on the reading behavior of programmers must carefully select source code snippets to control the influence of confounding factors. Furthermore, we identify a need for further studies on how programmers should structure source code to align it with their natural reading behavior to ease program comprehension.

More Detailed Suggestions

- Detailed suggestions exist
- For a summary: “Reporting Experiments in Software Engineering” [1]
- Generally, follow conventions/suggestions of the audience (thesis advisor, journal, ...)

Table 1. Characterization of different proposal of guidelines for empirical SE

	Singer [14]	Wohlin et al. [25]	Kitchenham et al. [6]	Juristo and Moreno [26]	Kitchenham [3]	New Proposal
Title	*	*	*	*	Title	Title
Authorship	*	*	*	*	Authorship	Authorship
Keywords		*	*	*	Keywords	Keywords
Type of Study	Empirical Research	Empirical Research	Empirical Research	Controlled Experiment	Systematic Review	Controlled Experiment
Phases of Study	Reporting	All	All	All	All	Reporting
Structure	Abstract	*	*	*	Executive Summary or Structured Abstract	Structured Abstract
	Introduction	Introduction	*	Goal Definition	Background	Introduction
		Problem Statement				
		Experiment Planning	Experimental Context			
	Introduction	Problem Statement	Experimental Context	Goal Definition	Background	Related Work
	Method	Experiment Planning	Experimental Design	Design	Review Questions Review Methods	Experiment Planning
	Procedure	Experiment Operation	Conducting the Experiment and Data Collection	Experiment Execution	Included and Excluded Studies	Execution
	Results	Data Analysis	Analysis	Experimental Analysis	Results	Analysis
	Discussion	Interpretation of Results	Interpretation of Results	Experimental Analysis	Discussion	Discussion
	Discussion	Discussion and Conclusion	*	Experimental Analysis	Conclusion	Conclusions & Future Work
					Acknowledgments Conflict of Interest	Acknowledgements
	References	References	*	*	References	References
	Appendices	Appendix			Appendices	Appendices

Too Much Effort for Practitioners?

„I am working as a software developer. I don't have time for detailed documentation! I'll give just a quick PowerPoint presentation of the results.“

- → fatal mistake that can make the entire effort pointless
- If you do not have time to write it down, you probably did not have time to properly and carefully conduct all steps of the experiment
- Verbal communication has a much higher loss of information and is not permanent
- Written communication forces neutrality

[1] Jedlitschka, Ciolkowski, Pfahl "[Reporting Experiments in Software Engineering](#)"

Hints and Fine-Tuning: Writing



- Ideally, along with the report, everything relevant is also published
 - Raw data and analysis scripts for an controlled experiment (→ replications!)
 - Executable software code for tool demonstrations
 - Nowadays, it is often required for published papers and reviewed
- There are several platforms that offer long-term archival
 - Figshare, OSF, Zenodo
 - Do not use GitHub (alone)



- Writing is tough
- But, it can be learned with some practice
 - Be clear and concise
 - Use simple language (e.g., utilize -> use)
 - Academic writing breaks a lot of the rules you may have learned in school (e.g., synonyms)
- Writing is part of research and of paramount importance
 - Does not refer to grammatical or spelling mistakes (they should never happen anyway with today's tools)
 - Argument structure, sentence structure, ... should be polished and requires as much work as the experiment

- One of the most common problems for students that can ruin great work
- Use the direct resources that are out there
 - Check prior and related work
 - Check the writing and presentation of “best paper” awards from related top conferences
 - Ask advisor/supervisor for examples (they will be delighted you asked)
- Do not get too attached to your writing
 - If your writing/report is criticized, it is not a personal attack. Understand it as objective feedback to improve

- Recommended general writing resources from my experience
 - <http://www.inf.fu-berlin.de/lehre/pmo/eng/ScientificWriting.pdf>
 - <https://www.coursera.org/learn/sciwrite/>
 - http://www.dansimons.com/resources/Simons_on_writing_1.5.pdf
 - Recommended book: „Science Research Writing For Non-Native Speakers Of English“, Hilary Glasman-Deal
- Caution: be careful with writing resources
 - They can cause writer's block, especially if consumed in a large amount

“A good dissertation is a done dissertation. A great dissertation is a published dissertation. A perfect dissertation is neither.”

- In summary, make sure to get the structure of the report right
- Clearly communicate the message
 - Reviewers may not agree with it, but understanding your viewpoint is essential
- Ideally, the report follows a “story”
 - Present this story to a fellow student, colleague or a patient parent
 - What do they say? What do they ask? Do you need additional arguments?
- Title, abstract, introduction and conclusion get the most attention
 - Make sure to polish those before anything else

Writing: Digits after Comma

- Many students copy & paste statistical values into their report
- “We invited 32 participants with an average age of 27.923151 years”
- “We invited 32 participants with an average age of 27.9 years”

Writing: Digits after Comma

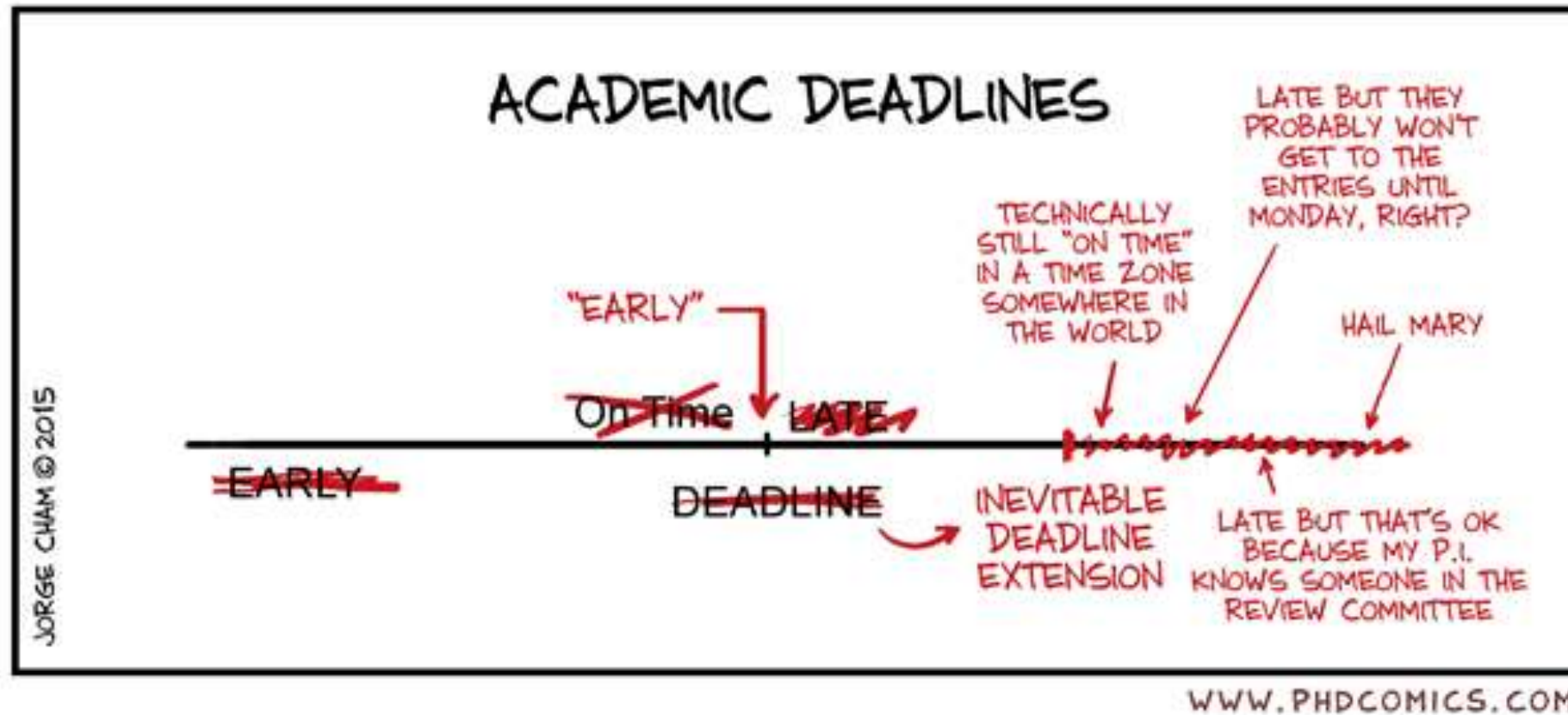
- Many students copy & paste statistical values into their report
- ✗ “We invited 32 participants with an average age of 27.923151 years”
 - The last digit indicates age in 30s increments. Is the data really that accurate?
- ✓ “We invited 32 participants with an average age of 27.9 years”

Pearson Product Moment Correlation – Audi and BMW		
Statistic	Variable X (Audi)	Variable Y (BMW)
Mean	73.9264705882353	66.3382352941177
Biased Variance	105.391652249135	68.6282439446367
Biased Standard Deviation	10.2660436512385	8.2842165558752
Covariance	1.15780334273931	
Correlation	0.0135637768510468	
Determination	0.000183976042464994	
T-Test	0.222896101102961	
p-value (2 sided)	0.823784901589857	
p-value (1 sided)	0.411892450794928	
Kendall tau Rank Correlation		
Kendall tau	-0.0187808685004711	
2-sided p-value	0.657646954059601	
Score	-660	
Var(Score)	2211268.5	
Denominator	35142.14453125	

Table 16 – Absolute Sentiment Score – Correlation between Audi and BWM

Writing: Start Early

- Start well ahead of the deadline



Writing: Start Early

- Start well ahead of the deadline

Norman Peitek / Untitled project / normans-awesome-dissertation

Commits

Search commits

All branches


Author	Commit	Message	Date
Norman Peitek	04872d7	minor update to reflect current status	2018-06-29
Norman Peitek	9996ce7	add references for two EMIP, ESEM, and TSE papers	2018-05-09
Norman Peitek	28feb1f	mention some updates	2018-04-06
Norman Peitek	f58aea1	no message	2018-02-09
Norman Peitek	7132bf9	one more attempt to fix header/footer	2018-02-02
Norman Peitek	0898b59	make header/footer consistent add draft version to footer	2018-02-02
Norman Peitek	038edb4	continue to outline dissertation	2018-02-02
Norman Peitek	3dbfba9	minor design cleanups, structuring	2018-01-26
Norman Peitek	b87e609	extend gitignore	2018-01-26
Norman Peitek	9e61204	cleanup repo: remove unused files	2018-01-26
Norman Peitek	9933e2d	remove pdf file from git	2018-01-26
Norman Peitek	f519fe1	start to structure dissertation	2018-01-26
Norman Peitek	aebe3fc	initial commit	2017-03-08

WRITING YOUR THESIS OUTLINE

NOTHING
SPOUSE

STEP 1 Aim for a respectable number of chapters:

THESIS OUTLINE

- 1.
- 2.
- 3.
- 4.
5.  chapter #'s
- 6.
- 7.

5 = "That's IT??"
6-7 = "Not bad"
8+ = "Are you crazy??"

STEP 2 Fill in the "freebies":

THESIS OUTLINE

1. INTRODUCTION
2. LIT REVIEW
3. METHODOLOGY
- 4.
- 5.
- 6.
7. CONCLUSIONS

You're half way done!

Hints and Fine-Tuning: Visualization



- Summary statistics are not enough (remember Anscombe's quartet)
- Use the right plot for the right use case
 - ~~Never~~ *I guess sometimes* use pie charts and try to avoid bar plots in scientific work

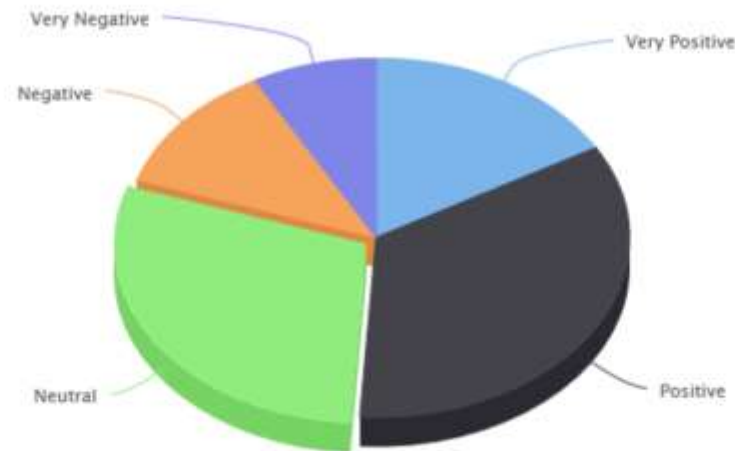
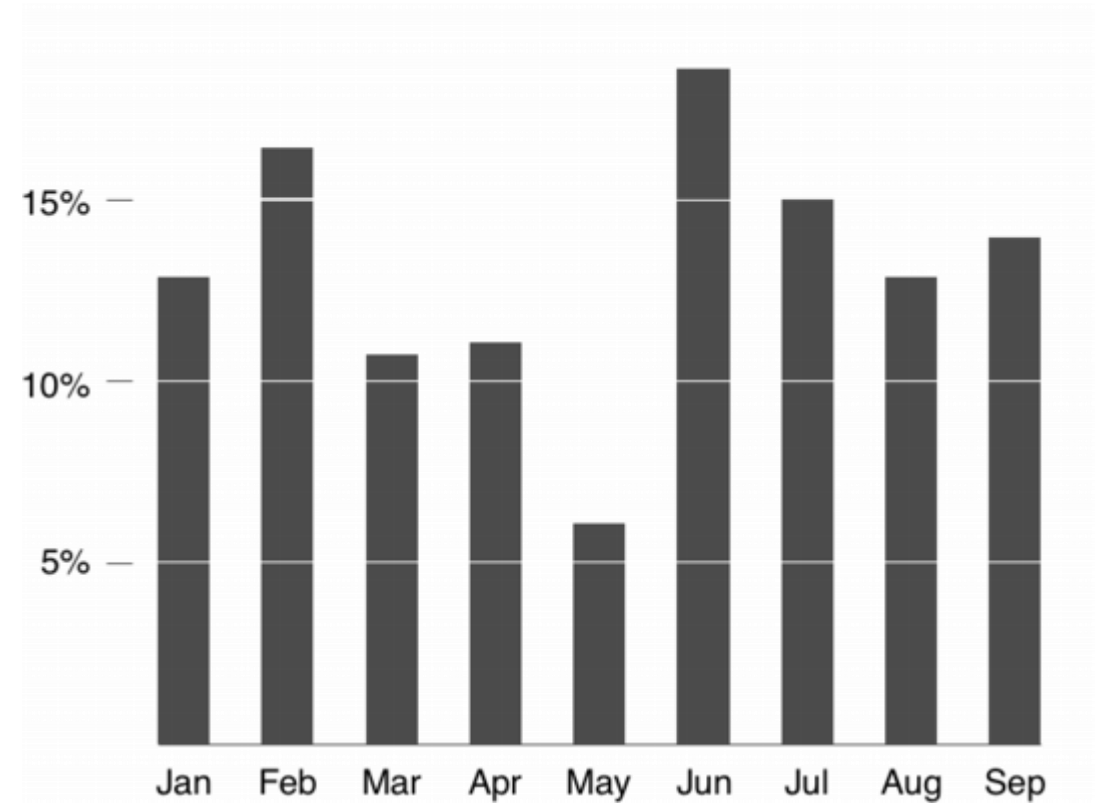
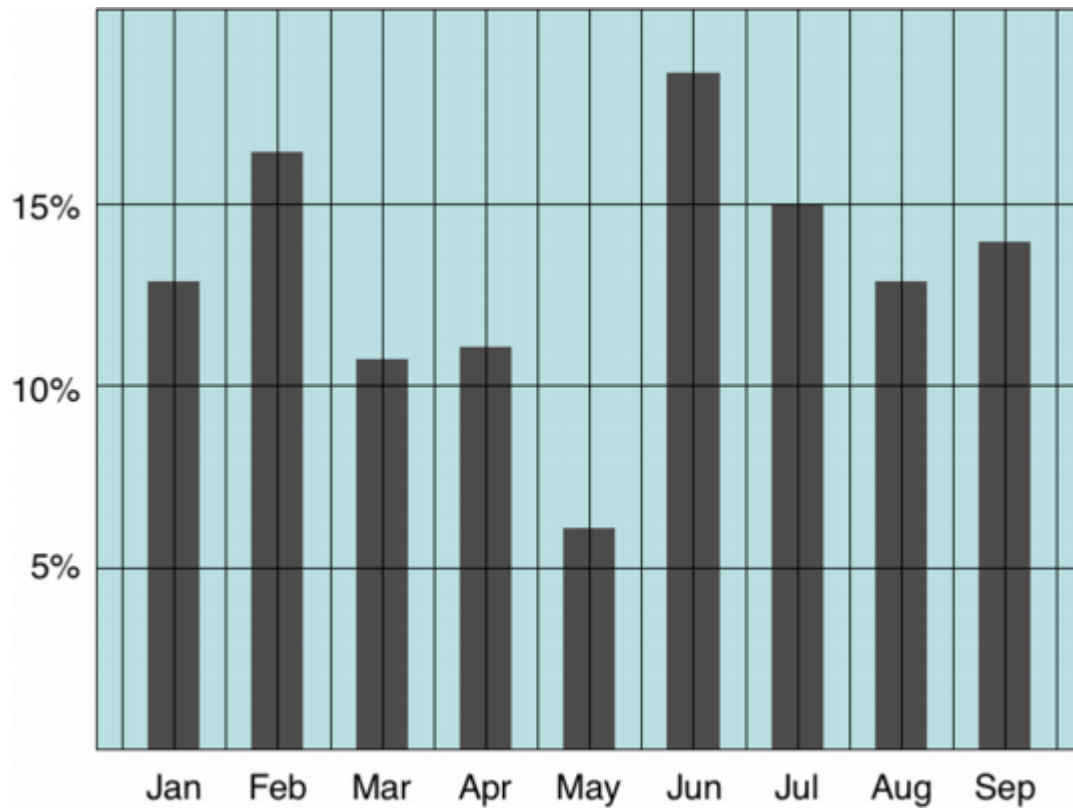


Figure 11 – Pie Chart of Tweets by Sentiment Score (Monthly Summary)

- In academia: Less ink is preferred
 - “Data-ink” ratio should be as high as possible
- Advantages
 - If the plot has little ink, it automatically highlights the important aspects
 - Saves space
- Process
 - Remove as much non-data ink as reasonably possible
 - Remove redundant data ink as much as possible

Visualization: Less Ink



https://infovis-wiki.net/wiki/Data-Ink_Ratio

Visualization: Less Ink for Tables

Short-term credit lines by bank

June 30, 1980 (millions USD)

Bank	Finance Companies					Manufacturing Companies							Grand Total
	Canada	Germany	UK	USA	Total	Argentina	Australia	Canada	France	UK	USA	Total	
Allied & Associates	0.0	0.0	18.6	0.0	18.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	18.6
Bank of America	0.0	0.0	0.0	17.5	17.5	0.0	3.3	0.0	0.0	0.0	17.5	20.8	38.3
Bankers Trust	0.0	0.0	0.0	13.0	13.0	0.0	0.0	0.0	0.0	0.0	13.0	13.0	26.0
Banque National de Paris	0.0	0.0	11.3	0.0	11.3	0.0	0.0	15.0	34.6	0.0	0.0	49.6	60.9
Barclays	0.0	0.0	42.5	0.0	42.5	0.0	0.0	0.0	0.0	133.4	0.0	133.4	175.9
Chase Manhattan	0.0	0.0	0.0	15.2	15.2	0.0	0.0	0.0	0.0	0.0	15.0	15.0	30.2
Chemical	0.0	0.0	0.0	13.3	13.3	0.0	0.0	0.0	0.0	0.0	13.0	13.0	26.3
CIBC	37.8	0.0	27.1	0.0	64.9	0.0	0.0	222.9	3.6	3.4	0.0	229.9	294.8
Citibank	0.0	0.0	0.0	17.5	17.5	0.5	0.0	0.0	0.0	1.1	17.5	19.1	36.6
Commerzbank	0.0	3.3	0.0	0.0	3.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.3
Continental Illinois	0.0	0.0	4.6	21.8	26.4	0.0	0.0	0.0	0.0	0.0	21.1	21.1	47.5
Crédit Lyonnais	0.0	0.0	9.0	0.0	9.0	0.0	0.0	0.0	33.0	0.0	3.0	36.0	45.0
Deutsche Bank	0.0	3.3	0.0	0.0	3.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.3
Dresdner	0.0	3.3	0.0	0.0	3.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.3
FNB Chicago	0.0	0.0	0.0	15.0	15.0	0.0	0.0	0.0	0.0	0.0	15.0	15.0	30.0
Lloyds	0.0	0.0	42.5	0.0	42.5	0.0	0.0	0.0	0.0	36.2	0.0	36.2	78.7
Midland	0.0	0.0	54.3	0.0	54.3	0.0	0.0	0.0	0.0	36.2	0.0	36.2	90.5
Royal Bank of Canada	0.0	0.0	11.3	0.0	11.3	0.0	0.0	0.0	0.0	0.0	15.0	15.0	26.3
Société General	0.0	0.0	9.0	0.0	9.0	0.0	0.0	0.0	50.1	0.0	0.0	50.1	59.1
Toronto Dominion	0.0	0.0	6.8	0.0	6.8	0.0	0.0	0.0	0.0	0.0	15.2	15.2	22.0
Others	0.0	7.8	50.8	78.1	136.7	40.1	25.5	0.0	16.1	19.2	48.2	149.1	285.8
Total	37.8	17.7	287.8	191.4	534.7	40.6	28.8	237.9	137.4	229.5	193.5	867.7	1,402.4

Short-term credit lines by bank

June 30, 1980 (millions USD)

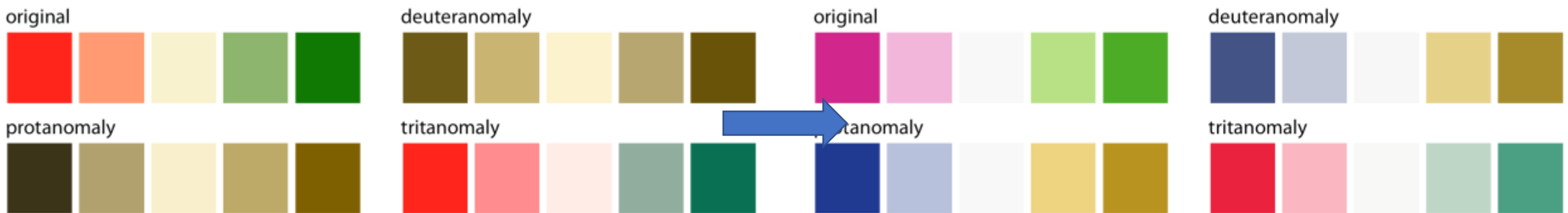
Bank	Grand Total	MANUFACTURING COMPANIES							FINANCE COMPANIES				
		Canada	UK	USA	France	Argentina	Australia	Total	UK	USA	Canada	Germany	Total
Total	1,402.4	237.9	229.5	193.5	137.4	40.6	28.8	867.7	287.8	191.4	37.8	17.7	534.7
CIBC	294.8	222.9	3.4	-	3.6	-	-	229.9	27.1	-	37.8	-	64.9
Others	285.8	-	19.2	48.2	16.1	40.1	25.5	149.1	50.8	78.1	-	7.8	136.7
Barclays	175.9	-	133.4	-	-	-	-	133.4	42.5	-	-	-	42.5
Midland	90.5	-	36.2	-	-	-	-	36.2	54.3	-	-	-	54.3
Lloyds	78.7	-	36.2	-	-	-	-	36.2	42.5	-	-	-	42.5
Banque National de Paris	60.9	15.0	-	-	34.6	-	-	49.6	11.3	-	-	-	11.3
Société General	59.1	-	-	-	50.1	-	-	50.1	9.0	-	-	-	9.0
Continental Illinois	47.5	-	-	21.1	-	-	-	21.1	4.6	21.8	-	-	26.4
Crédit Lyonnais	45.0	-	-	3.0	33.0	-	-	36.0	9.0	-	-	-	9.0
Bank of America	38.3	-	-	17.5	-	-	3.3	20.8	-	17.5	-	-	17.5
Citibank	36.6	-	1.1	17.5	-	0.5	-	19.1	-	17.5	-	-	17.5
Chase Manhattan	30.2	-	-	15.0	-	-	-	15.0	-	15.2	-	-	15.2
FNB Chicago	30.0	-	-	15.0	-	-	-	15.0	-	15.0	-	-	15.0
Royal Bank of Canada	26.3	-	-	15.0	-	-	-	15.0	11.3	-	-	-	11.3
Chemical	26.3	-	-	13.0	-	-	-	13.0	-	13.3	-	-	13.3
Bankers Trust	26.0	-	-	13.0	-	-	-	13.0	-	13.0	-	-	13.0
Toronto Dominion	22.0	-	-	15.2	-	-	-	15.2	6.8	-	-	-	6.8
Allied & Associates	18.6	-	-	-	-	-	-	-	18.6	-	-	-	18.6
Commerzbank	3.3	-	-	-	-	-	-	-	-	-	-	3.3	3.3
Deutsche Bank	3.3	-	-	-	-	-	-	-	-	-	-	3.3	3.3
Dresdner	3.3	-	-	-	-	-	-	-	-	-	-	3.3	3.3

<https://simplexct.com/data-ink-ratio-tables>

- Vector graphics (.svg, .pdf) are much preferred over bitmap graphics (.png)
 - Only exception: photos (equipment, ...)
- Vector-based files are much smaller and scale, which means much higher quality
- Most plotting libraries have the functionality to export is pdf/svg build in

Visualization: Color Scheme

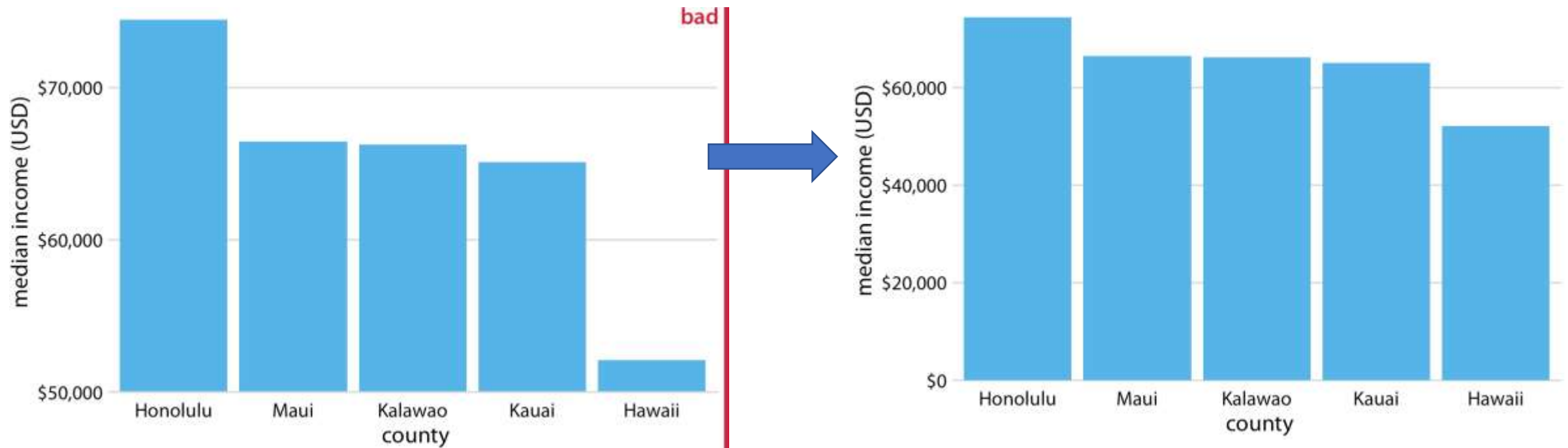
- Many default color schemes do not account for color deficiencies
 - Something in the range of 5-10% of men have color deficiencies



- Pick good color schemes: <https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3>

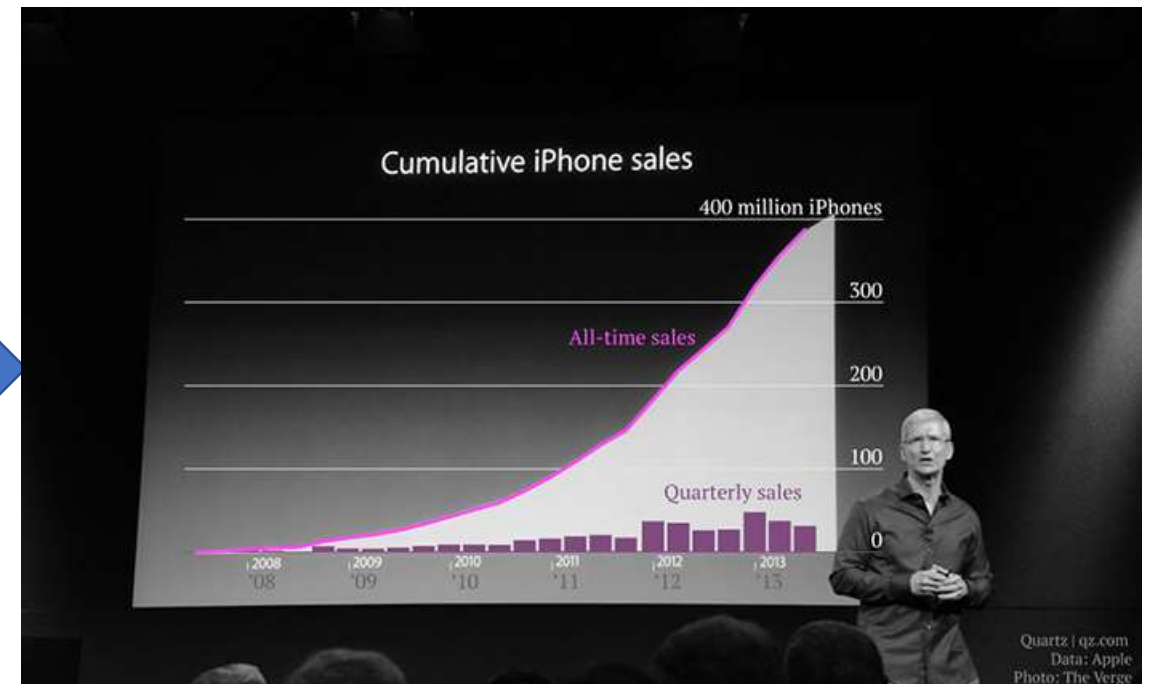
Visualization: Misleading Plots

- Do not present data in a misleading way



Visualization: Misleading Plots

- Do not present data in a misleading way



<https://www.techjunkie.com/tim-cook-trying-prove-meaningless-chart/>

- <https://www.cedricscherer.com/slides/USGS-1921-beyond-bar-and-box-plots.pdf>
- <https://colorbrewer2.org/#>
- <https://www.color-blindness.com/coblis-color-blindness-simulator/>
- <https://colororacle.org/>