Freie Universität Berlin

Erste Schritte zu einer Taxonomie von Software Engineering Kontexten

Zwischenpräsentation zur Masterarbeit

Stephanie Hohenberg, 5293431
hohenberg_s@web.de
Freie Univerisät Berlin

Prof. Dr. Lutz Prechelt
Freie Universität Berlin

Agenda

Motivation & Ziele

Hintergrund

Recherchefragen

Vorgehen & Zwischenstände

Motivationsbeispiel

©Thekla Barck / temel-art

Das ist Thekla.

Thekla arbeitet in der Software Engineering Arbeitsgruppe ihrer Universität und recherchiert für ihre Forschungsfrage.

Sie findet ganz viele verwandte Paper von Konferenzen mit interessanten Ergebnissen und fängt an ein Paper zu schreiben.

Motivationsbeispiel

©Thekla Barck / temel-art

Sie ist fast fertig mit ihren Paper.

Doch dann fällt auf, die Paper, auf die sie sich bezieht, haben andere Kontexte und stimmen nicht mit ihren Kontext überein.

Ziele

1. Terminologie zur Beschreibung vom Kontext einer Arbeit
   - Verbesserung von Kommunikation
   - Vereinfachte Einordnung eines Papers in die SE Welt

2. Förderung der Übertragbarkeit von Ergebnis
   - Bessere Einschätzung der Relevanz einer Arbeit

3. Verständnis der ICSE 2020
   - Vorgehen: Literaturstudie
   - Verständnis zu den aktuellen Beiträge im SE
   - Verständnis zu Trends und Lücken

Hintergrund

Qualitätsfaktoren empirischer Arbeiten

- Glaubwürdigkeit, Gründlichkeit – Internal Validity
- Relevanz, Verallgemeinbarkeit – External Validty

In wieweit wird der Kontext einer Arbeit hinsichtlich dessen Anwendbarkeit und Verallgemeinbarkeit beschrieben?

Hintergrund

Was ist überhaupt ein Kontext?

Ein Kontext definiert den Bereich, in dem die Ergebnisse einer Arbeit anwendbar sind.

Annahme: Kontexte kommen an verschiedenen Stellen in einem Paper vor, abhängig von ihrem Zweck

Vorgehen: Systematisches Suchen

Zweck	Vorkommen
Thema	Titel, Einleitung, Recherchefragen
Detail	Beschreibung, Vorgehen, Datenerhebung
Abgrenzung	Diskussion, verwandte Arbeiten
Verallgemeinbarkeit	Schlussfolgerungen, Forschungsbeiträge, Diskussion

Recherchefragen

RQ1

- Kontexte

RQ2

- Diskussionen zur Verallgemeinbarkeit
- Kontexte zum Zweck der Verallgemeinbarkeit

RQ3

- Korrelation: Kontexte <--> Arten der Publikationen bzw. Vorgehensweise der empirischen Studien

Recherchefragen

RQ1

- Welche Kontexte werden in den Publikationen erwähnt? In wie weit beschreiben Autoren und Autorinnen den Kontext ihrer Publikationen?
- In welcher Form kommen Kontexte vor? Sind diese explizit oder implizit erkennbar?
- Sind andere Zwecke als, die bereits genannten, erkennbar?

RQ2

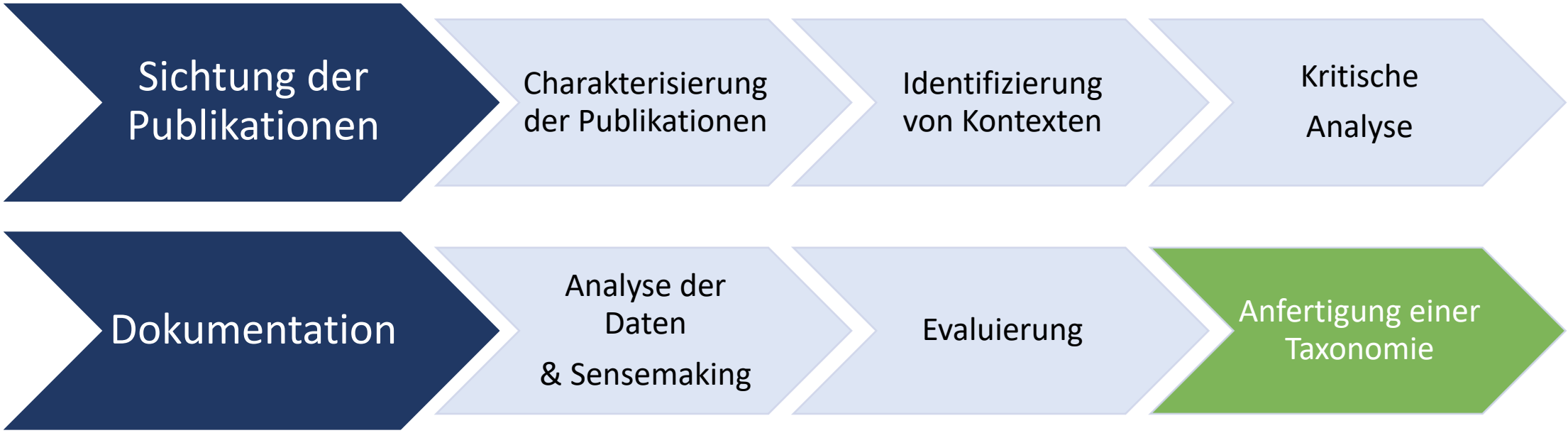
- In wie weit diskutieren Autoren und Autorinnen über die Verallgemeinbarkeit ihr Publikationen, Ergebnisse und Techniken?
- Bestehen Lücken in den Diskussionen zur Verallgemeinbarkeit?
- Wie unterscheiden sich die Kontexte zum Zweck der Verallgemeinbarkeit zu den Kontexte anderer Zwecke?

RQ3

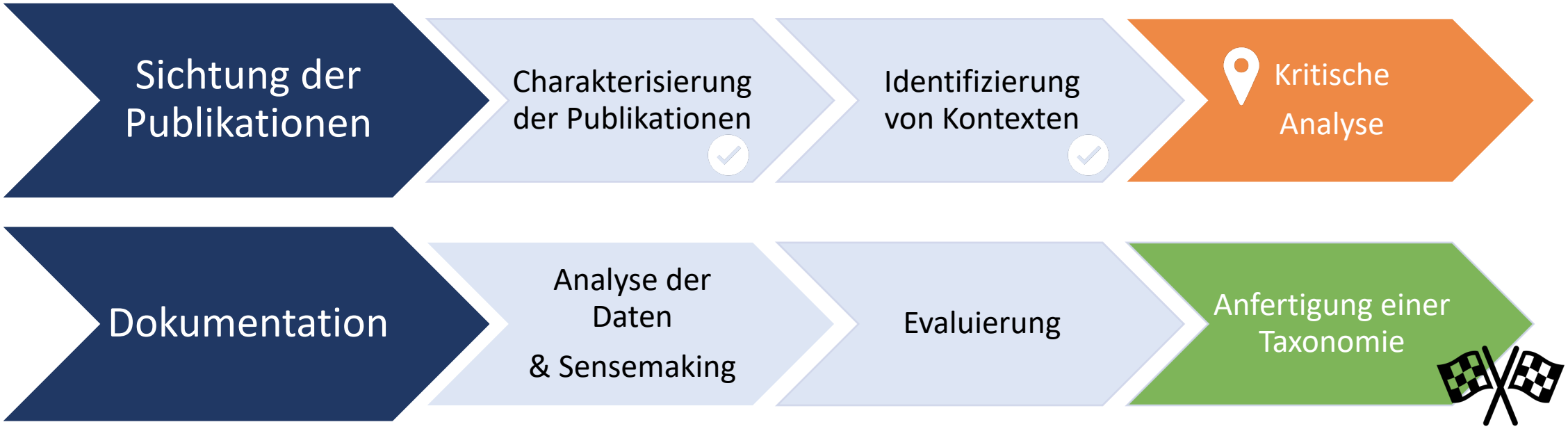
- Unterscheiden sich die Kontexte abhängig von den Arten der Publikationen und von der Vorgehensweise der empirischen Studien?

Vorgehen

Vorgehen

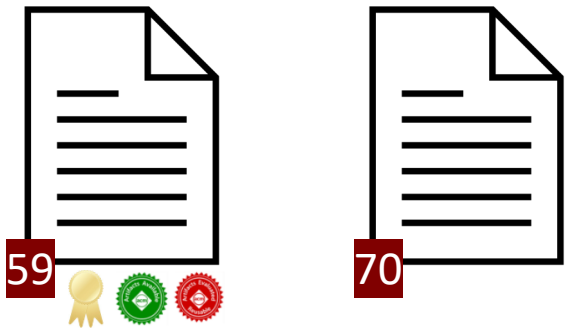

Sichtung der Publikationen
Charakterisierung der Publikationen
Identifizierung von Kontexten
Kritische Analyse
Dokumentation
Analyse der Daten & Sensemaking
Evaluierung
Anfertigung einer Taxonomie

Zwischenstand



Vorgehen

Sichtung der Publikationen



Sichtung der technischen Paper der ICSE 2020

Fokus: Publikationen mit Auszeichnung oder Badge (59)

Annahme: Publikationen mit Auszeichnungen weisen eine hörere Qualität auf, und sollten …

… mehr über Kontexte diskutieren

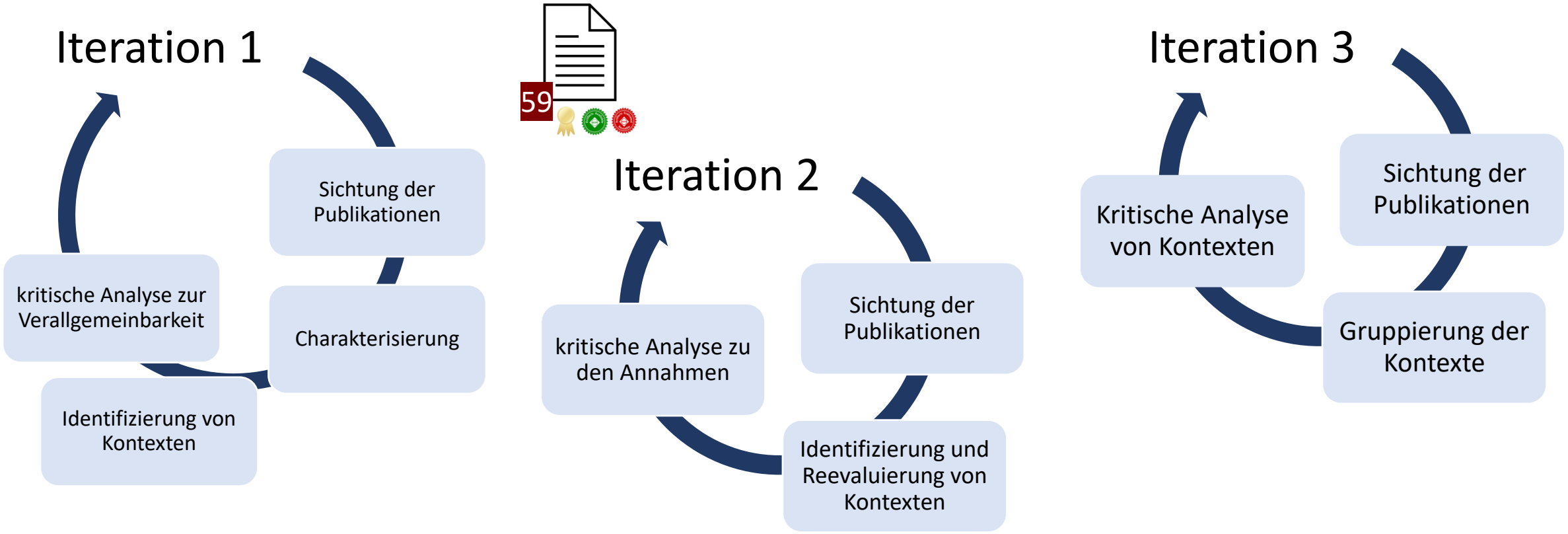
… mehr über die Verallgemeinbarkeit der Ergebnisse diskutieren

Vorgehen

Sichtung in 3 Iterationen

- mit unterschiedl. Clustering der Paper, unterschiedl. Lesereihenfolge
- mit Fokus auf
  - unterschiedl. Recherchefragen
  - unterschiedl. Stellen/Sektionen in den Paper
  - unterschiedl. Kontexte

Vorgehen


Iteration 1
Sichtung der Publikationen
Charakterisierung
Identifizierung von Kontexten
kritische Analyse zur Verallgemeinbarkeit
59
Iteration 2
Sichtung der Publikationen
Identifizierung und Reevaluierung von Kontexten
kritische Analyse zu den Annahmen
Iteration 3
Sichtung der Publikationen
Gruppierung der Kontexte
Kritische Analyse von Kontexten
Dokumentation

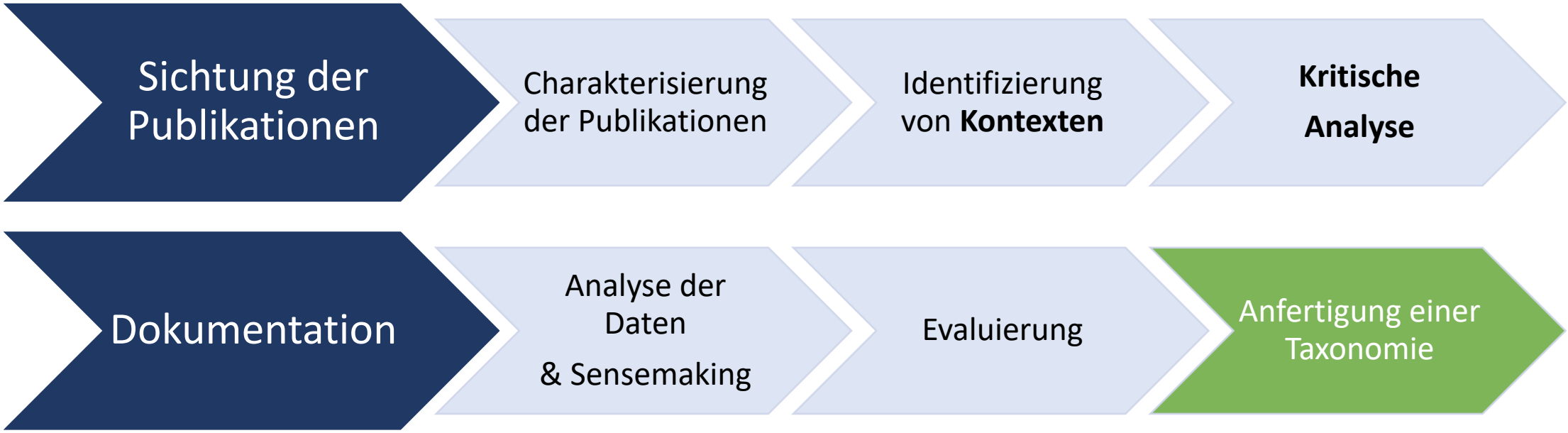
Vorgehen

Unterstützung durch Tools:

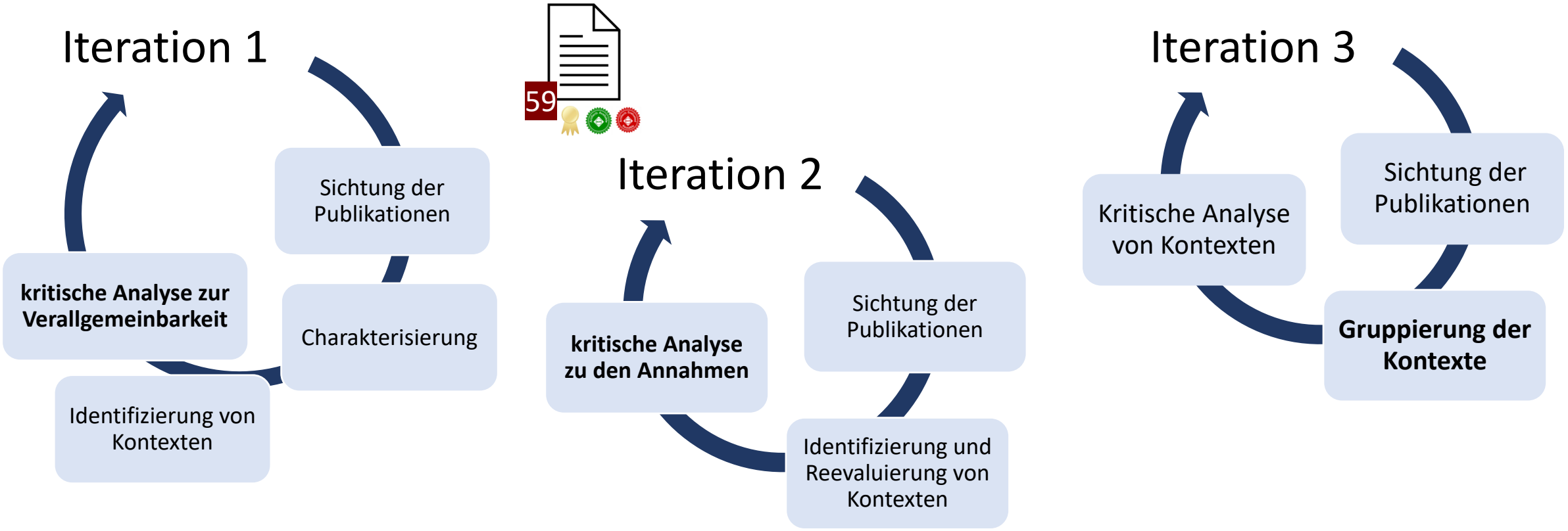
Entwicklung eines eigenen Tools

andere Tools:

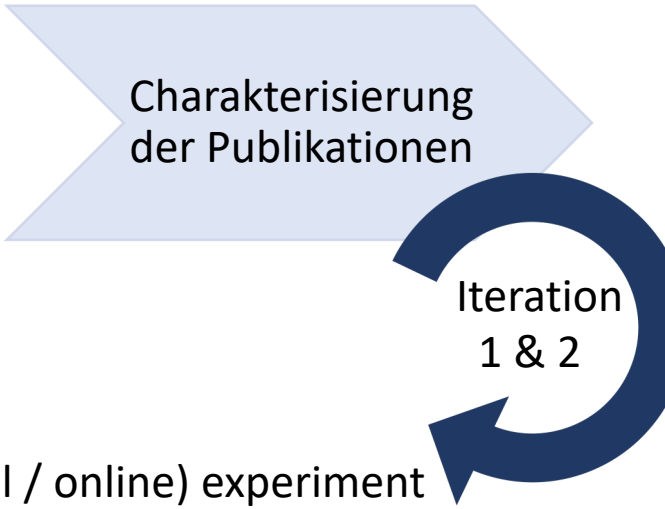
Vorgehen


Sichtung der Publikationen
Charakterisierung der Publikationen
Identifizierung von **Kontexten**
Kritische Analyse
Dokumentation
Analyse der Daten & Sensemaking
Evaluierung
Anfertigung einer Taxonomie

Vorgehen


Iteration 1
Sichtung der Publikationen
Charakterisierung
Identifizierung von Kontexten
kritische Analyse zur Verallgemeinbarkeit
59
Iteration 2
Sichtung der Publikationen
Identifizierung und Reevaluierung von Kontexten
kritische Analyse zu den Annahmen
Iteration 3
Sichtung der Publikationen
Gruppierung der Kontexte
Kritische Analyse von Kontexten
Dokumentation

Vorgehen – Iteration 1

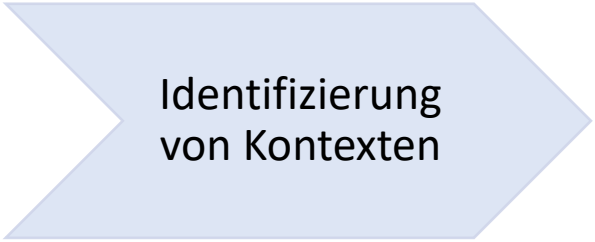


- Proposal: method / technique / approach (71,1%)
- Inspection, manuelle Analyse (32,2%)
- Empirical study (20,3%)
- Comprehensive study (11,8%)
- Pilot study (5,08%)
- Field study (1,6%)
- Exploratory study (6,7%)
- Case study (3,3%)
- User study (5,08%)

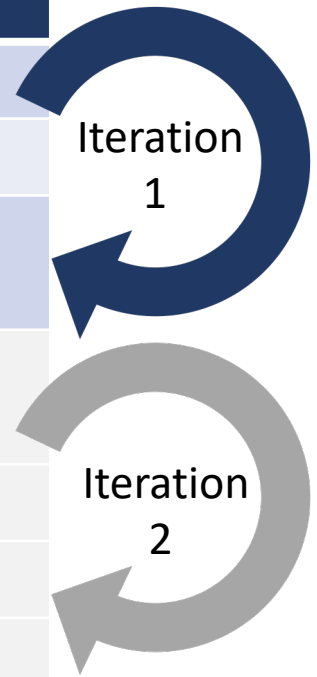
- Qualitative analysis (15,2%)
- Benchmark (33,8%)
- (controlled / psychologogical / online) experiment (38,9%)
- Simulation (1,6%)
- (semi-structured / structured) interview (11,8%)
- Survey (10,1%)
- Contextual inquiry (1,6%)
- States-of-arts / comparison (35,5%)

Themengebiete:
DL / NN (27,1%), Testing (35,5%), Data (x %), Human Aspects (13,5%)

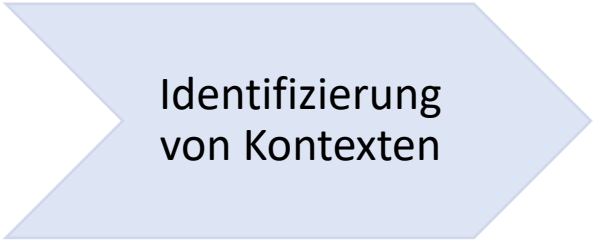
Vorgehen



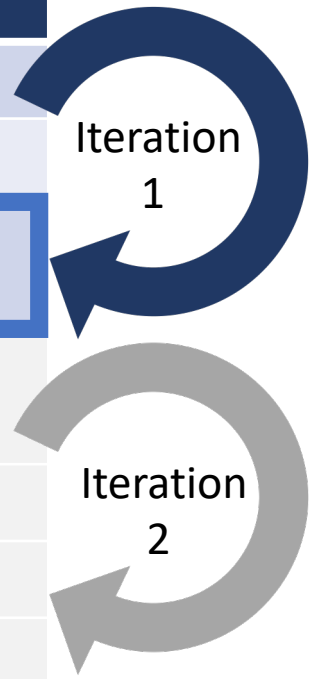
Zweck	Vorkommen
Thema	Titel, Abstract, Introduction
Ein-/Abgrenzung	Related Work
Verallgemeinbarkeit	Conclusions, Threats to Validity (extern validity), Discussion, Limitations, Future Work
Mögliche Verallgemeinbarkeit	Conclusion, Threats to Validity (extern validity), Discussion
Anwendung	Introduction, Vorgehen, Interpretation, Datenerhebung, Conclusion
Erweitbarkeit	Conclusion, Future Work
Details	Introduction, Vorgehen, Interpretation, Datenerhebung



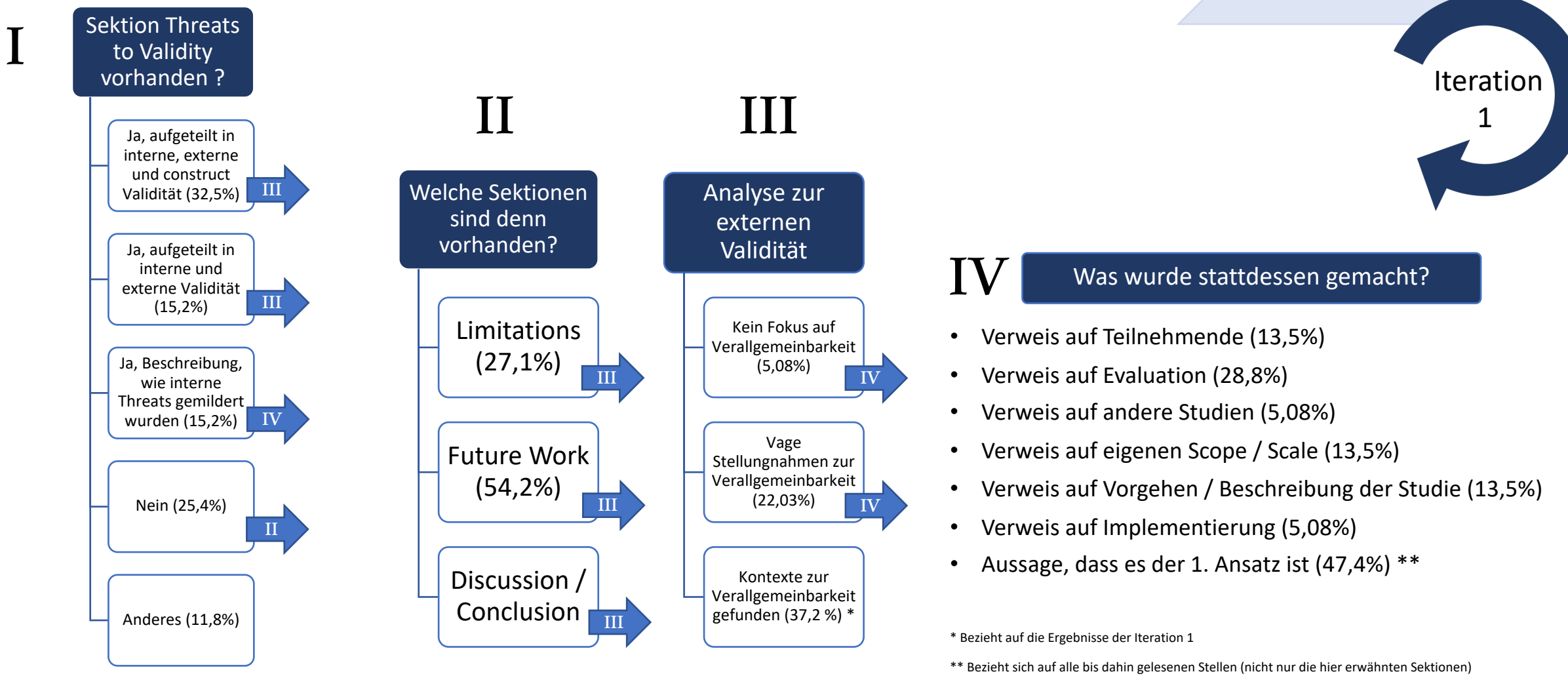
Vorgehen



Zweck	Vorkommen
Thema	Titel, Abstract, Introduction
Ein-/Abgrenzung	Related Work
Verallgemeinbarkeit	Conclusions, Threats to Validity (extern validity), Discussion, Limitations, Future Work
Mögliche Verallgemeinbarkeit	Conclusion, Threats to Validity (extern validity), Discussion
Anwendung	Introduction, Vorgehen, Interpretation, Datenerhebung, Conclusion
Erweitbarkeit	Conclusion, Future Work
Details	Introduction, Vorgehen, Interpretation, Datenerhebung



Vorgehen

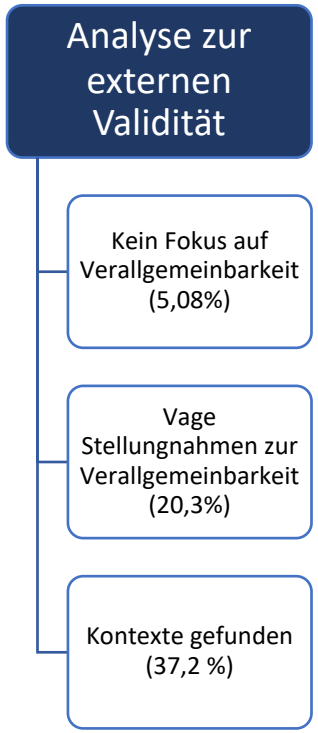


* Bezieht auf die Ergebnisse der Iteration 1

** Bezieht sich auf alle bis dahin gelesenen Stellen (nicht nur die hier erwähnten Sektionen)

Beispiele

Kritische Analyse zur Verallgemeinbarkeit



"While desirable, generalizability was not the main focus of this study" [1]

- Liegt der Fokus bei Paper mit ähnlicher Charakteristik ebenfalls nicht auf die Verallgemeinbarkeit?

"The 12 applications in our study clearly **may not represent all real-world applications**; the 114 issues studied also **may not represent all constraint-related issues** in these applications; the 100 participants of our user-study from MTurk **may not represent all real-world users**. Overall, we have tried our best to conduct an unbiased study" [2]

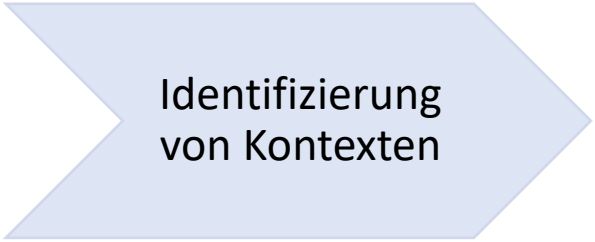
- Wieso nehmen die AutorInnen keine Stellung zur Verallgemeinbarkeit ein?
- Wieso beschreiben Sie nicht wenigstens ihren eigenen Kontext und dessen Grenzen?

Iteration 2

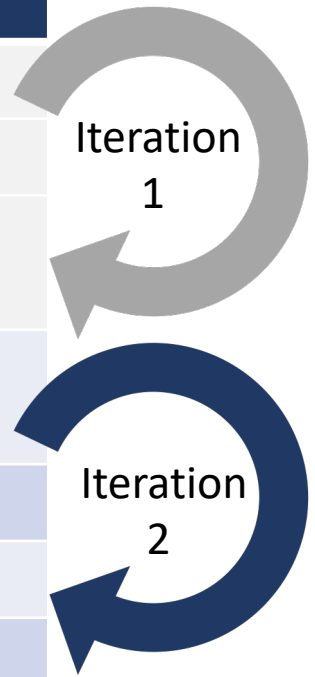
> Einführung des Zwecks: mögliche Verallgemeinbarkeit

> Vermutungen zur Verallgemeinbarkeit, Evaluierung der Annahmen

Vorgehen – Iteration 2



Zweck	Vorkommen
Thema	Titel, Abstract, Introduction
Ein-/Abgrenzung	Related Work
Verallgemeinbarkeit	Conclusions, Threats to Validity (extern validity), Discussion, Limitations, Future Work
Mögliche Verallgemeinbarkeit	Conclusion, Threats to Validity (extern validity), Discussion
Anwendung	Introduction, Vorgehen, Interpretation, Datenerhebung, Conclusion
Erweitbarkeit	Conclusion, Future Work
Details	Introduction, Vorgehen, Interpretation, Datenerhebung



Vorgehen

Kritische Analyse zu den Annahmen

Stellen:

Vorgehen, Interpretation, Datenerhebung, Conclusion, Threats to Validity, Discussion, Limitations, Future Work

Welche Annahmen machen die AutorInnen? Sind diese stimmig oder eher vage?

Beeinflussen diese Annahmen den Kontext der Arbeit oder dessen Relevanz?

+ tiefere Analyse / sensemaking zur Verallgemeinbarkeit

Clustering der Befunde bzgl in Annahmen in 6 Gruppen

bzgl eigene Einsichten in 6 Gruppen

Beispiele

Kritische Analyse zu den Annahmen

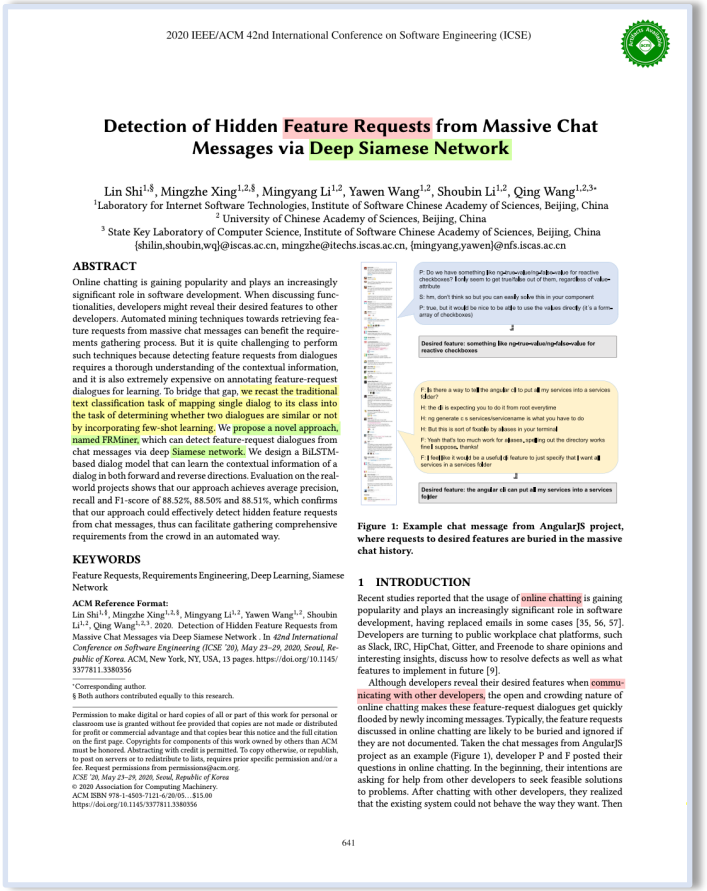
A641: Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network [3]

Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Shoubin Li, Qing Wang

PROPOSAL: approach | EVALUATION | STATES_OF_ARTS

DL_NN: Deep Siamase Network, bilSTM

- Vorstellung eines Ansatz zur Aufdeckung von Feature Request in Dialogen mittels eines Deep Siames Network
- Annotation von mehr als 1000 Dialogen von 3 OS Projekten zur Erstellung eines NN
- Effektivität und Verallgemeinbarkeit wird evaluiert



2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)

Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network

Lin Shi[1,§], Mingzhe Xing[1,2,§], Mingyang Li[1,2], Yawen Wang[1,2], Shoubin Li[1,2], Qing Wang[1,2,3,*]
[1]Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China
{shilin,shoubin,wq}@iscas.ac.cn, mingzhe@itechs.iscas.ac.cn, {mingyang,yawen}@nfs.iscas.ac.cn

ABSTRACT

Online chatting is gaining popularity and plays an increasingly significant role in software development. When discussing functionalities, developers might reveal their desired features to other developers. Automated mining techniques towards retrieving feature requests from massive chat messages can benefit the requirements gathering process. But it is quite challenging to perform such techniques because detecting feature requests from dialogues requires a thorough understanding of the contextual information, and it is also extremely expensive on annotating feature-request dialogues for learning. To bridge that gap, we recast the traditional text classification task of mapping single dialog to its class into the task of determining whether two dialogues are similar or not by incorporating few-shot learning. We propose a novel approach, named FRMiner, which can detect feature-request dialogues from chat messages via deep Siamese network. We design a BiLSTM-based dialog model that can learn the contextual information of a dialog in both forward and reverse directions. Evaluation on the real-world projects shows that our approach achieves average precision, recall and F1-score of 88.52%, 88.50% and 88.51%, which confirms that our approach could effectively detect hidden feature requests from chat messages, thus can facilitate gathering comprehensive requirements from the crowd in an automated way.

KEYWORDS

Feature Requests, Requirements Engineering, Deep Learning, Siamese Network

ACM Reference Format:
Lin Shi[1,§], Mingzhe Xing[1,2,§], Mingyang Li[1,2], Yawen Wang[1,2], Shoubin Li[1,2], Qing Wang[1,2,3]. 2020. Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network . In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380356

*Corresponding author.
§ Both authors contributed equally to this research.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380356

Figure 1: Example chat message from AngularJS project, where requests to desired features are buried in the massive chat history.

1 INTRODUCTION

Recent studies reported that the usage of online chatting is gaining popularity and plays an increasingly significant role in software development, having replaced emails in some cases [35, 56, 57]. Developers are turning to public workplace chat platforms, such as Slack, IRC, HipChat, Gitter, and Freenode to share opinions and interesting insights, discuss how to resolve defects as well as what features to implement in future [9].

Although developers reveal their desired features when communicating with other developers, the open and crowding nature of online chatting makes these feature-request dialogues get quickly flooded by newly incoming messages. Typically, the feature requests discussed in online chatting are likely to be buried and ignored if they are not documented. Taken the chat messages from AngularJS project as an example (Figure 1), developer P and F posted their questions in online chatting. In the beginning, their intentions are asking for help from other developers to seek feasible solutions to problems. After chatting with other developers, they realized that the existing system could not behave the way they want. Then

641

Beispiele

Kritische Analyse zu den Annahmen

A641: Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network [3]

Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Shoubin Li, Qing Wang

Einordnung: **AC1: stimmige Annahmen und Verallgemeinbarkeitsaussagen**

„We consider that dialogues expressing feature-requests share common linguistic patterns across domains that are typically not relevant with domain-specific concepts. The results show that FRMiner can learn these common patterns and be generalized to other projects.“

Einordnung: **AC2.1 vage/implizite Annahmen bzgl Kontexte zu Details und Anwendung**

- Keine Betrachtung von anderen Kommunikationswegen (Telefonate, persönlichen Gesprächen, …)
- keine Diskussion über mögliche Unterschiede zu industriellen/closed source Systemen
  - CS Projekte und OS Projekte kommunizieren anders

Beispiele

Kritische Analyse zu den Annahmen

B171: Primers or Reminders? The Effects of Existing Review Comments on Code Review [4]

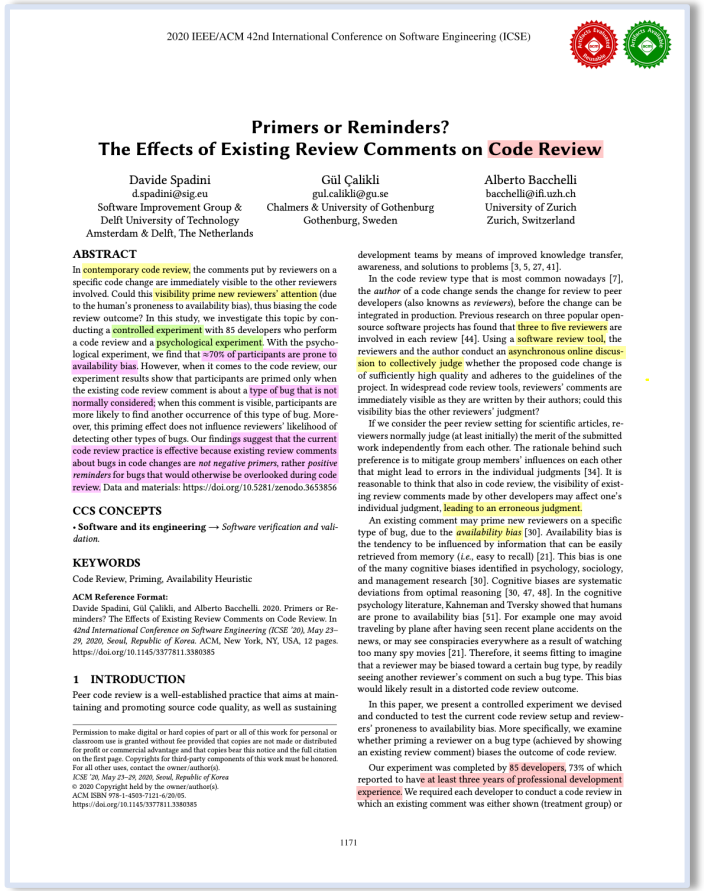
Davide Spadini, Gül Çalikli, Alberto Bacchelli

EXPERIMENT | CONTROLLED_EXPERIMENT | PSYCHOLOGICAL_EXPERIMENT

ONLINE_EXPERIMEN | SURVEY: questionnaire after the experiment

HUMAN_ASPECTS | QUALITATIVE_ANALYSIS

- Durchführung von Experimenten zur Untersuchung, ob bestehende Kommentare über Bugs einen Einfluss in der kollaborativen Code Review hat
- Experimente mit Java Programmen und 3 vers Bugtypen



2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)

Primers or Reminders?
The Effects of Existing Review Comments on Code Review

Davide Spadini
d.spadini@sig.eu
Software Improvement Group &
Delft University of Technology
Amsterdam & Delft, The Netherlands

Gül Çalikli
gul.calikli@gu.se
Chalmers & University of Gothenburg
Gothenburg, Sweden

Alberto Bacchelli
bacchelli@ifi.uzh.ch
University of Zurich
Zurich, Switzerland

ABSTRACT

In contemporary code review, the comments put by reviewers on a specific code change are immediately visible to the other reviewers involved. Could this visibility prime new reviewers' attention (due to the human's proneness to availability bias), thus biasing the code review outcome? In this study, we investigate this topic by conducting a controlled experiment with 85 developers who perform a code review and a psychological experiment. With the psychological experiment, we find that ≈70% of participants are prone to availability bias. However, when it comes to the code review, our experiment results show that participants are primed only when the existing code review comment is about a type of bug that is not normally considered; when this comment is visible, participants are more likely to find another occurrence of this type of bug. Moreover, this priming effect does not influence reviewers' likelihood of detecting other types of bugs. Our findings suggest that the current code review practice is effective because existing review comments about bugs in code changes are *not negative primers*, rather *positive reminders* for bugs that would otherwise be overlooked during code review. Data and materials: https://doi.org/10.5281/zenodo.3653856

CCS CONCEPTS

• **Software and its engineering** → *Software verification and validation.*

KEYWORDS

Code Review, Priming, Availability Heuristic

ACM Reference Format:
Davide Spadini, Gül Çalikli, and Alberto Bacchelli. 2020. Primers or Reminders? The Effects of Existing Review Comments on Code Review. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380385

1 INTRODUCTION

Peer code review is a well-established practice that aims at maintaining and promoting source code quality, as well as sustaining development teams by means of improved knowledge transfer, awareness, and solutions to problems [3, 5, 27, 41].

In the code review type that is most common nowadays [7], the *author* of a code change sends the change for review to peer developers (also knowns as *reviewers*), before the change can be integrated in production. Previous research on three popular open-source software projects has found that three to five reviewers are involved in each review [44]. Using a software review tool, the reviewers and the author conduct an asynchronous online discussion to collectively judge whether the proposed code change is of sufficiently high quality and adheres to the guidelines of the project. In widespread code review tools, reviewers' comments are immediately visible as they are written by their authors; could this visibility bias the other reviewers' judgment?

If we consider the peer review setting for scientific articles, reviewers normally judge (at least initially) the merit of the submitted work independently from each other. The rationale behind such preference is to mitigate group members' influences on each other that might lead to errors in the individual judgments [34]. It is reasonable to think that also in code review, the visibility of existing review comments made by other developers may affect one's individual judgment, leading to an erroneous judgment.

An existing comment may prime new reviewers on a specific type of bug, due to the *availability bias* [30]. Availability bias is the tendency to be influenced by information that can be easily retrieved from memory (*i.e.*, easy to recall) [21]. This bias is one of the many cognitive biases identified in psychology, sociology, and management research [30]. Cognitive biases are systematic deviations from optimal reasoning [30, 47, 48]. In the cognitive psychology literature, Kahneman and Tversky showed that humans are prone to availability bias [51]. For example one may avoid traveling by plane after having seen recent plane accidents on the news, or may see conspiracies everywhere as a result of watching too many spy movies [21]. Therefore, it seems fitting to imagine that a reviewer may be biased toward a certain bug type, by readily seeing another reviewer's comment on such a bug type. This bias would likely result in a distorted code review outcome.

In this paper, we present a controlled experiment we devised and conducted to test the current code review setup and reviewers' proneness to availability bias. More specifically, we examine whether priming a reviewer on a bug type (achieved by showing an existing review comment) biases the outcome of code review.

Our experiment was completed by 85 developers, 73% of which reported to have at least three years of professional development experience. We required each developer to conduct a code review in which an existing comment was either shown (treatment group) or

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05.
https://doi.org/10.1145/3377811.3380385

Beispiele

Kritische Analyse zu den Annahmen

B171: Primers or Reminders? The Effects of Existing Review Comments on Code Review [4]

Davide Spadini, Gül Çalikli, Alberto Bacchelli

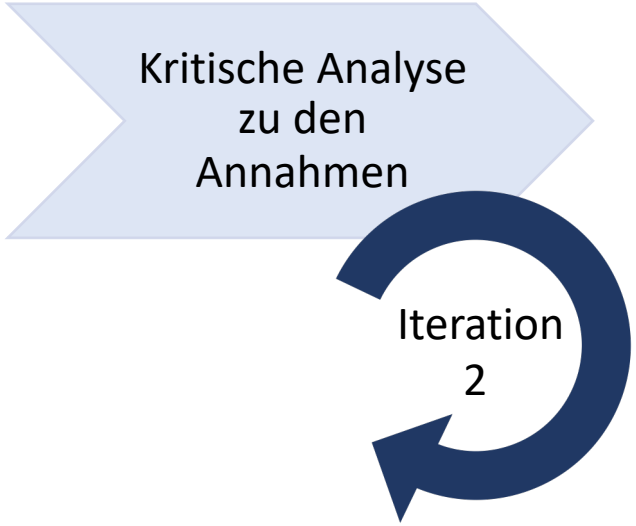
Einordnung: **AC2.3 implizite Annahmen, die Relevanz gefährden**

- Annahme, dass in der Industrie kolloborativ Code gereviewt
- Annahme, dass jemand mehrere Bugs auf einmal commitet
- Untersuchung von sehr trivialen Bugs (return-of-a-wrong-value Bugs, Null Pointer Exception), die durch Tests abgedeckt werden sollten

Einordnung: **AC2.2 Annahmen bzgl Vorgehen, Kritik**

- Keine Simulation vom Requirements Engineering (bzgl. Corner Case Bugs)

Vorgehen



Annahmen

AC1 stimmige Annahmen und Aussagen zur Verallgemeinbarkeit

AC2.1 vage/implizite Annahmen bzgl Kontexte ohne genügend Begründung

AC2.2 Annahmen bzgl Vorgehen, Kritik

AC2.3 implizite Annahmen, die die Relevanz des Publikation gefährden

AC3 Annahme, Verallgemeinbarkeit abhängig von Trainingsmodell

AC4 keine Annahmen gefunden (vermutlich wegen Charakteristik der Publikation)

Einsichten

IC1 zu wenig Expertenwissen für tiefere Analyse

IC2 unstimmige Argumentation

IC3.1 weniger Verallgemeinbarkeitsaussagen + Verweis auf Future Work / neues Recherchefeld / mehr Forschungsbedarf

IC3.2 weniger Verallgemeinbarkeitsaussagen + first approach / exploratory / qualitative study

IC3.3 weniger Verallgemeinbarkeitsaussagen + Verweis auf prior work

IC4 implizite Kontexte entdeckt

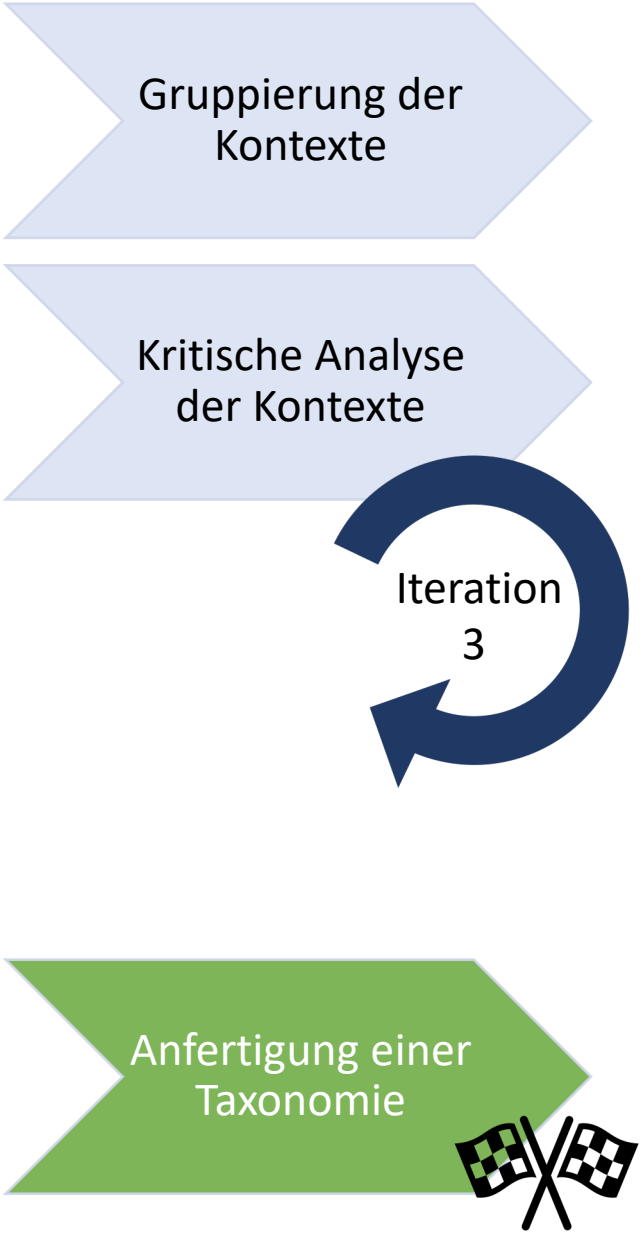
Vorgehen – Iteration 3

Paper übergreifend bezogen auf:

- Anwendung
- Programmiersprache
- Projekt
- Human-related
- SE Aktivität
- Sourcing

Paper bezogen:

Bilden die Kontexte einer Publikation bereits einen Baum, welcher für die Taxonomie verwendet werden kann?



Beispiele

Gruppierung der Kontexte

Paper-übegreifende Kontexte zur Software-Engineering Aktivitäten

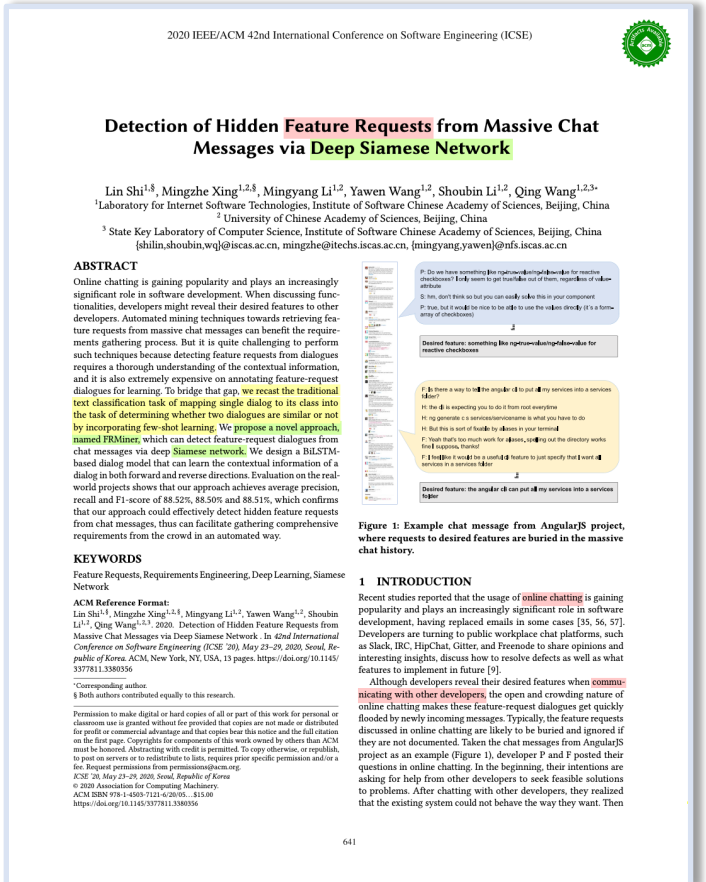
Beispiele

Gruppierung der Kontexte

A641: Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network [3]

Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Shoubin Li, Qing Wang

- **Anwendung:** (issue tracking system)
- **Programmiersprache:** /
- **Projekt:**
  - cross-project settings
  - Projekte mit online chatting
  - englisch sprachige Projekte
- **Human-related:** Kommunikation
- **SE Aktivität:** Requirements Engineering
- **Sourcing:** open, (closed)



2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)

Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network

Lin Shi[1,§], Mingzhe Xing[1,2,§], Mingyang Li[1,2], Yawen Wang[1,2], Shoubin Li[1,2], Qing Wang[1,2,3*]
[1]Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China
{shilin,shoubin,wq}@iscas.ac.cn, mingzhe@itechs.iscas.ac.cn, {mingyang,yawen}@nfs.iscas.ac.cn

ABSTRACT

Online chatting is gaining popularity and plays an increasingly significant role in software development. When discussing functionalities, developers might reveal their desired features to other developers. Automated mining techniques towards retrieving feature requests from massive chat messages can benefit the requirements gathering process. But it is quite challenging to perform such techniques because detecting feature requests from dialogues requires a thorough understanding of the contextual information, and it is also extremely expensive on annotating feature-request dialogues for learning. To bridge that gap, we recast the traditional text classification task of mapping single dialog to its class into the task of determining whether two dialogues are similar or not by incorporating few-shot learning. We propose a novel approach, named FRMiner, which can detect feature-request dialogues from chat messages via deep Siamese network. We design a BiLSTM-based dialog model that can learn the contextual information of a dialog in both forward and reverse directions. Evaluation on the real-world projects shows that our approach achieves average precision, recall and F1-score of 88.52%, 88.50% and 88.51%, which confirms that our approach could effectively detect hidden feature requests from chat messages, thus can facilitate gathering comprehensive requirements from the crowd in an automated way.

KEYWORDS

Feature Requests, Requirements Engineering, Deep Learning, Siamese Network

ACM Reference Format:
Lin Shi[1,§], Mingzhe Xing[1,2,§], Mingyang Li[1,2], Yawen Wang[1,2], Shoubin Li[1,2], Qing Wang[1,2,3*]. 2020. Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network . In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380356

*Corresponding author.
§ Both authors contributed equally to this research.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380356

Figure 1: Example chat message from AngularJS project, where requests to desired features are buried in the massive chat history.

1 INTRODUCTION

Recent studies reported that the usage of online chatting is gaining popularity and plays an increasingly significant role in software development, having replaced emails in some cases [35, 56, 57]. Developers are turning to public workplace chat platforms, such as Slack, IRC, HipChat, Gitter, and Freenode to share opinions and interesting insights, discuss how to resolve defects as well as what features to implement in future [9].

Although developers reveal their desired features when communicating with other developers, the open and crowding nature of online chatting makes these feature-request dialogues get quickly flooded by newly incoming messages. Typically, the feature requests discussed in online chatting are likely to be buried and ignored if they are not documented. Taken the chat messages from AngularJS project as an example (Figure 1), developer P and F posted their questions in online chatting. In the beginning, their intentions are asking for help from other developers to seek feasible solutions to problems. After chatting with other developers, they realized that the existing system could not behave the way they want. Then

641

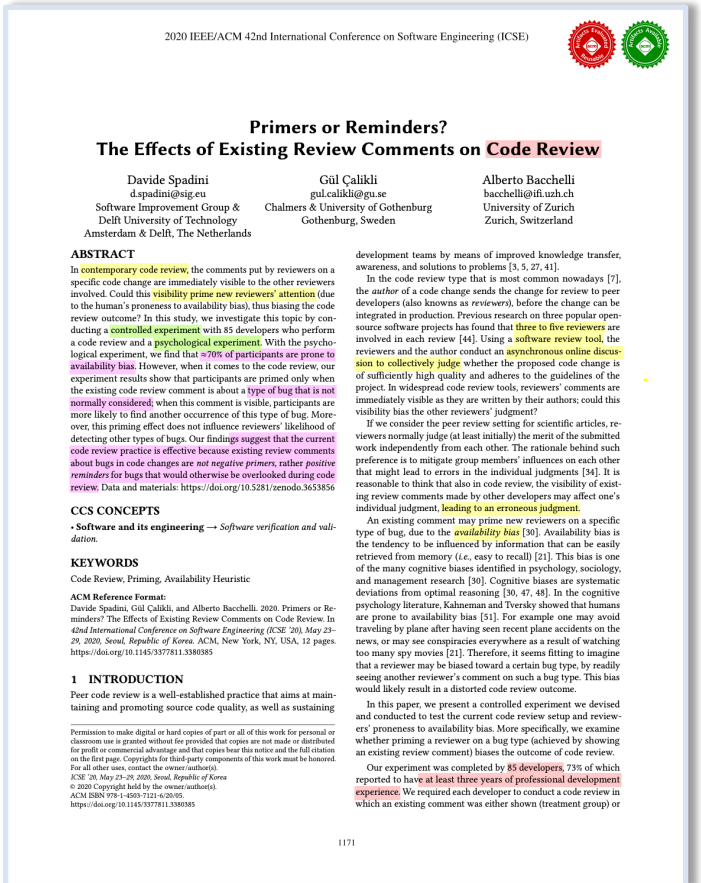
Beispiele

Gruppierung der Kontexte

B171: Primers or Reminders? The Effects of Existing Review Comments on Code Review [4]

Davide Spadini, Gül Çalikli, Alberto Bacchelli

- **Anwendung:** Java Programs
- **Programmiersprache:** Java
- **Projekt:**
  - Projekte, die Werzeuge zur Code Review mit Online Discussion Feature benutzen
  - große Projekte mit einem großen Dev Team
- **Human-related:** Rolle: EntwicklerInnen, Erfahrung der EntwicklerInnen
- **SE Aktivität:** Code Review
- **Sourcing:** closed



2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)

Primers or Reminders?
The Effects of Existing Review Comments on Code Review

Davide Spadini
d.spadini@sig.eu
Software Improvement Group & Delft University of Technology
Amsterdam & Delft, The Netherlands

Gül Çalikli
gul.calikli@gu.se
Chalmers & University of Gothenburg
Gothenburg, Sweden

Alberto Bacchelli
bacchelli@ifi.uzh.ch
University of Zurich
Zurich, Switzerland

ABSTRACT

In contemporary code review, the comments put by reviewers on a specific code change are immediately visible to the other reviewers involved. Could this visibility prime new reviewers' attention (due to the human's proneness to availability bias), thus biasing the code review outcome? In this study, we investigate this topic by conducting a controlled experiment with 85 developers who perform a code review and a psychological experiment. With the psychological experiment, we find that ≈70% of participants are prone to availability bias. However, when it comes to the code review, our experiment results show that participants are primed only when the existing code review comment is about a type of bug that is not normally considered; when this comment is visible, participants are more likely to find another occurrence of this type of bug. Moreover, this priming effect does not influence reviewers' likelihood of detecting other types of bugs. Our findings suggest that the current code review practice is effective because existing review comments about bugs in code changes are *not negative primers*, rather *positive reminders* for bugs that would otherwise be overlooked during code review. Data and materials: https://doi.org/10.5281/zenodo.3653856

CCS CONCEPTS

• **Software and its engineering** → *Software verification and validation.*

KEYWORDS

Code Review, Priming, Availability Heuristic

ACM Reference Format:
Davide Spadini, Gül Çalikli, and Alberto Bacchelli. 2020. Primers or Reminders? The Effects of Existing Review Comments on Code Review. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380385

1 INTRODUCTION

Peer code review is a well-established practice that aims at maintaining and promoting source code quality, as well as sustaining development teams by means of improved knowledge transfer, awareness, and solutions to problems [3, 5, 27, 41].

In the code review type that is most common nowadays [7], the *author* of a code change sends the change for review to peer developers (also knowns as *reviewers*), before the change can be integrated in production. Previous research on three popular open-source software projects has found that three to five reviewers are involved in each review [44]. Using a software review tool, the reviewers and the author conduct an asynchronous online discussion to collectively judge whether the proposed code change is of sufficiently high quality and adheres to the guidelines of the project. In widespread code review tools, reviewers' comments are immediately visible as they are written by their authors; could this visibility bias the other reviewers' judgment?

If we consider the peer review setting for scientific articles, reviewers normally judge (at least initially) the merit of the submitted work independently from each other. The rationale behind such preference is to mitigate group members' influences on each other that might lead to errors in the individual judgments [34]. It is reasonable to think that also in code review, the visibility of existing review comments made by other developers may affect one's individual judgment, leading to an erroneous judgment.

An existing comment may prime new reviewers on a specific type of bug, due to the *availability bias* [30]. Availability bias is the tendency to be influenced by information that can be easily retrieved from memory (*i.e.*, easy to recall) [21]. This bias is one of the many cognitive biases identified in psychology, sociology, and management research [30]. Cognitive biases are systematic deviations from optimal reasoning [30, 47, 48]. In the cognitive psychology literature, Kahneman and Tversky showed that humans are prone to availability bias [51]. For example one may avoid traveling by plane after having seen recent plane accidents on the news, or may see conspiracies everywhere as a result of watching too many spy movies [21]. Therefore, it seems fitting to imagine that a reviewer may be biased toward a certain bug type, by readily seeing another reviewer's comment on such a bug type. This bias would likely result in a distorted code review outcome.

In this paper, we present a controlled experiment we devised and conducted to test the current code review setup and reviewers' proneness to availability bias. More specifically, we examine whether priming a reviewer on a bug type (achieved by showing an existing review comment) biases the outcome of code review.

Our experiment was completed by 85 developers, 73% of which reported to have at least three years of professional development experience. We required each developer to conduct a code review in which an existing comment was either shown (treatment group) or

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

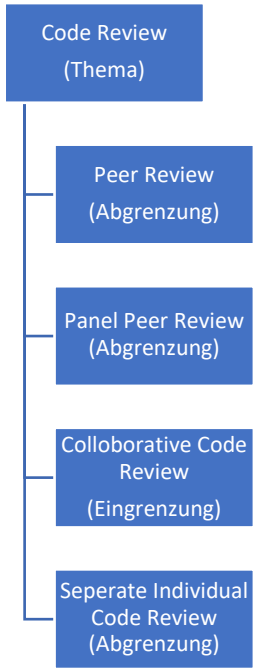
ACM ISBN 978-1-4503-7121-6/20/05.
https://doi.org/10.1145/3377811.3380385

Beispiele

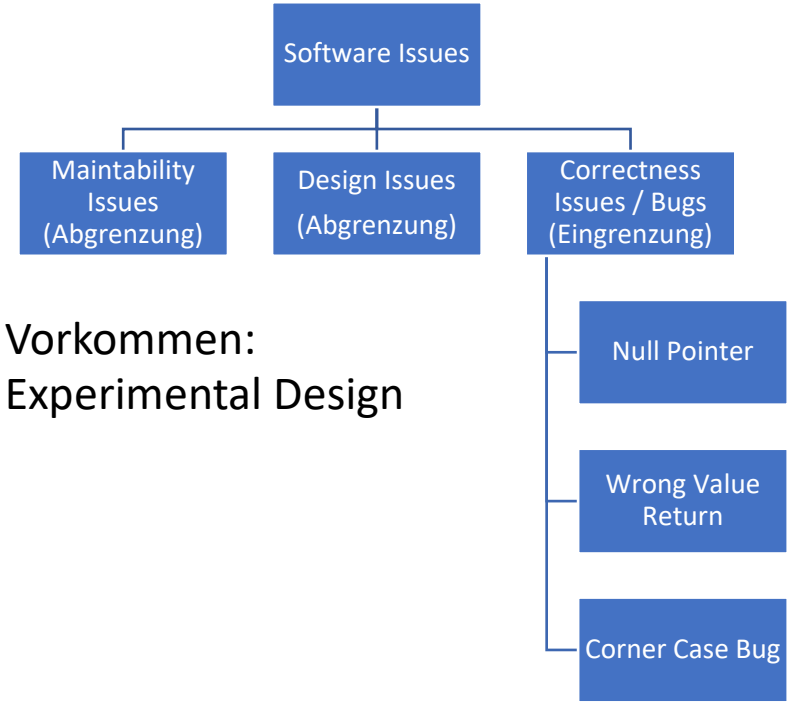
Gruppierung der Kontexte

B171: Primers or Reminders? The Effects of Existing Review Comments on Code Review [4]

Davide Spadini, Gül Çalikli, Alberto Bacchelli



Vorkommen:
Introduction
Related Work

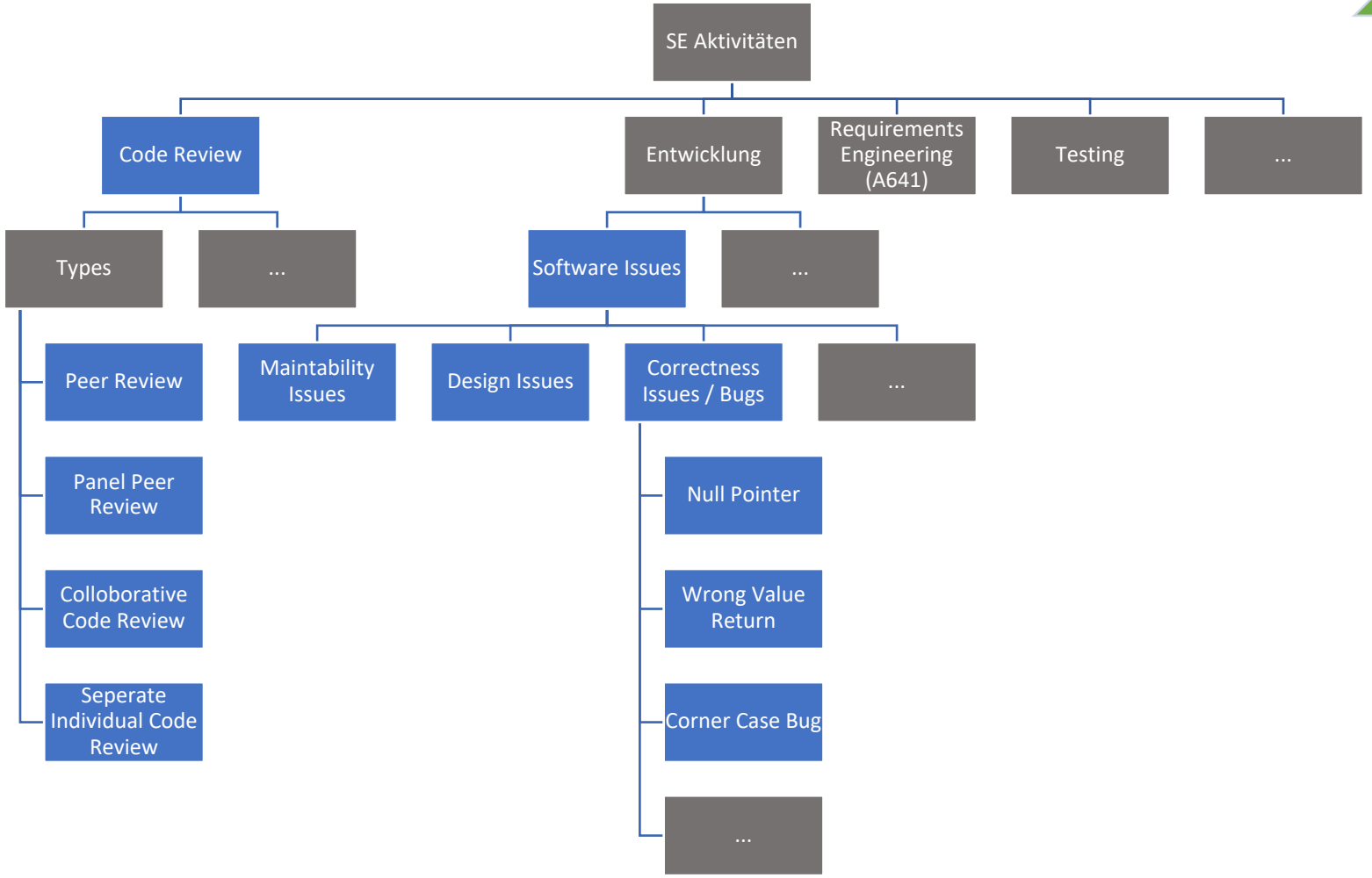


Vorkommen:
Experimental Design

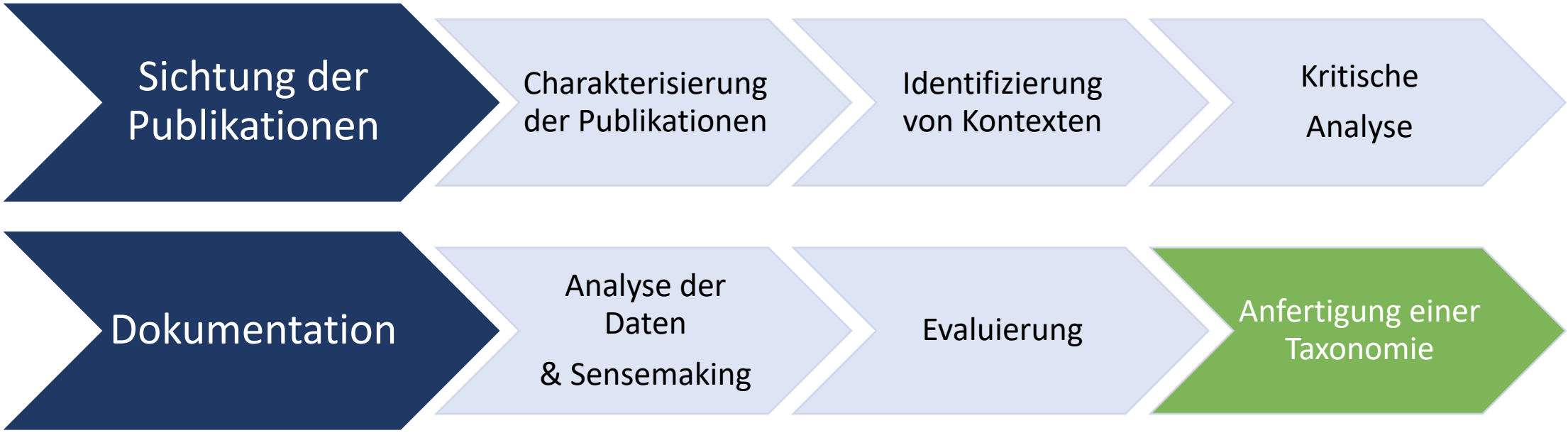
Anfertigung einer Taxonomie

Beispiele

Anfertigung einer Taxonomie



Vorgehen


Sichtung der Publikationen
Charakterisierung der Publikationen
Identifizierung von Kontexten
Kritische Analyse
Dokumentation
Analyse der Daten & Sensemaking
Evaluierung
Anfertigung einer Taxonomie

Vorgehen

Entwicklung eines Tools

- Use Cases:
  - Datenvisualisierung
  - Analyse
- Andere potentielle Use Cases:
  - Dokumentation
  - Versionierung der Daten

Zwischenstand: Tool

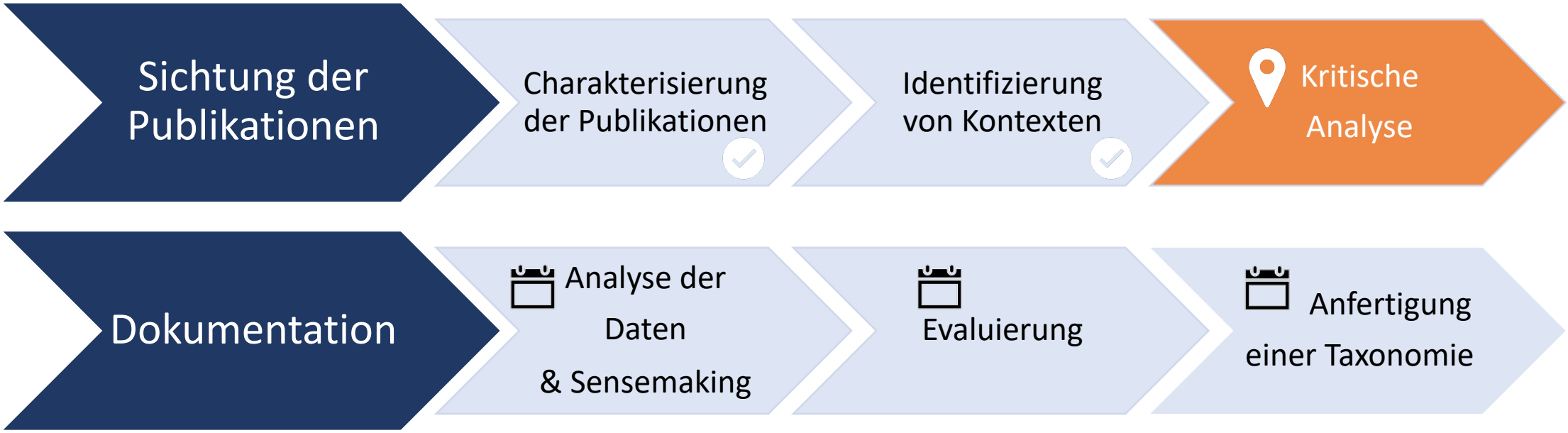
Bereits implementiert:

- Übersicht über die Publikationen und Daten
- Filterfunktion

Restliche TODOs:

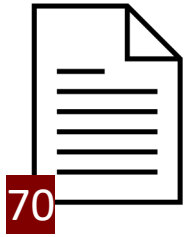
- Befüllung aller Daten
- Graphvisualisierung der Kontexte und der Taxonomie
- Diagramme zur Analyse der Daten
- Negativfilter

Zwischenstand



Weiteres Vorgehen

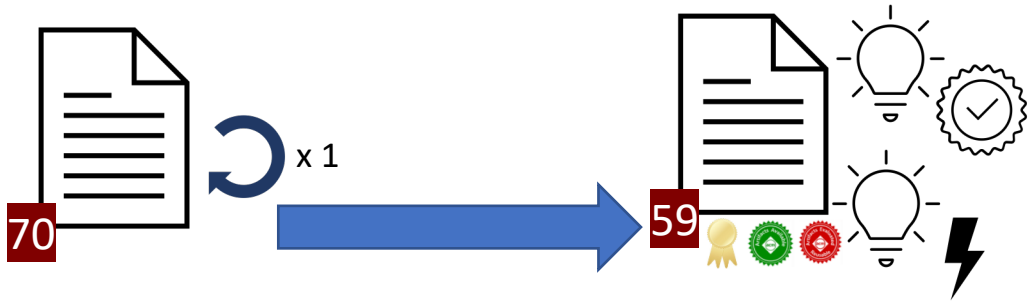
- Analyse der Daten & Sensemaking
- Beantwortung aller Recherchefragen
- Anfertigung der finalen Taxonomie
- Evaluation meines Vorgehens
- Sichtung der restlichen Publikationen
- Anfertigung der Masterarbeit



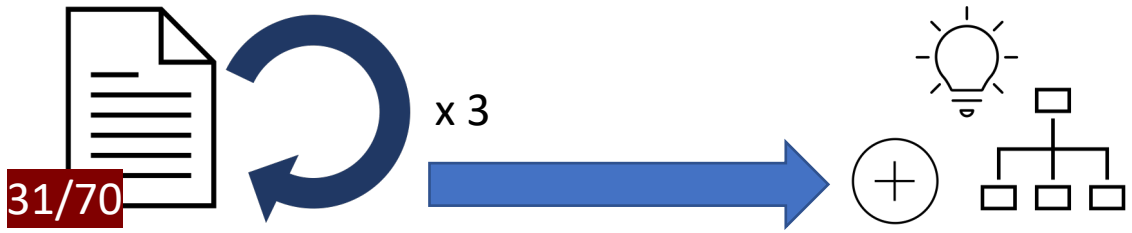
Weiteres Vorgehen

Sichtung der restlichen Paper ohne Auszeichnung oder Badge

Option 1:
- Charakterisierung
- Analyse zur Verallgemeinbarkeit
- Einordnung in die Taxonomie ohne neue Kontexte
- Evaluierung der Einsichten von der Sichtung der Paper mit Auszeichnung
- Mehr Sensemaking



Option 2:
- Auswahl der Paper und Clustering
- Iteration 1-3 wiederholen
- Erweiterung der Taxonomie
- Mehr Einsichten



Freie Universität Berlin

Erste Schritte zu einer Taxonomie von Software Engineering Kontexten

Zwischenpräsentation zur Masterarbeit

Diskussion:
Wir würden Sie weiter vorgehen?
Option 1 vs Option 2

Threats

1. Keine Forschungserfahrung, kein Expertenwissen
2. Keine Vorkenntnisse vom Material
3. Biase, beim Sichten der Publikationen (nach Interesse)
4. Übersehen von Kontexten durch Entwickeln einer Routine / Trotts
5. Englisch nicht als Muttersprache
6. Mehreres Wechseln der Tools zur Dokumentation
7. Konsistenzhaltung der Daten im Excel und im eigenen Tool

Freie Universität Berlin

Erste Schritte zu einer Taxonomie von Software Engineering Kontexten

Zwischenpräsentation zur Masterarbeit

Diskussionsfragen?
Anregungen?
Feedback?

Anhang

Referenzierte Publikationen

[1] A Tale from the Trenches: Cognitive Biases and Software Development

A654 - Souti Chattopadhyay, Nicholas Nelson, Audrey Au, Natalia Morales, Christopher Sanchez, Rahul Pandita, Anita Sarma

- Zwei-teilige Field Study mit 10 Teilnehmenden zur Untersuchung wie kognitive Biase die Softwareentwicklung beeinflusst
- Kontrollierte Experimente und Interviews

[2] Managing data constraints in database-backed web applications

B098 - Junwen Yang, Utsav Sethi, Cong , Alvin Cheung, Shan Lu

- Comprehensive Study zur Untersuchung von Data Constraints mittels einer manuellen Analyse von Source Code, Commit History und Issues Tracking System von 12 Anwendungen
- Vorstellung eines Tools zum Aufdecken von Data Violations und zur Erhöhen der Qualität von Anwendungen

Referenzierte Publikationen

[3] Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network A641 - Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Shoubin Li, Qing Wang

- Vorstellung eines Ansatz zur Aufdeckung von Feature Request in Dialogen mittels eines Deep Siames Network
- Annotation von mehr als 1000 Dialogen von 3 OS Projekten zur Erstellung eines NN
- Die Effektivität und die Verallgemeinbarkeit wird evaluiert.

[4] Primers or Reminders? The Effects of Existing Review Comments on Code Review B171 - Davide Spadini, Gül Çalikli, Alberto Bacchelli

- Durchführung von Experimenten zur Untersuchung, ob bestehende Kommentare über Bugs einen Einfluss in der kollaborativen Code Review hat
- Experimente mit Java Programmen und 3 vers Bugtypen