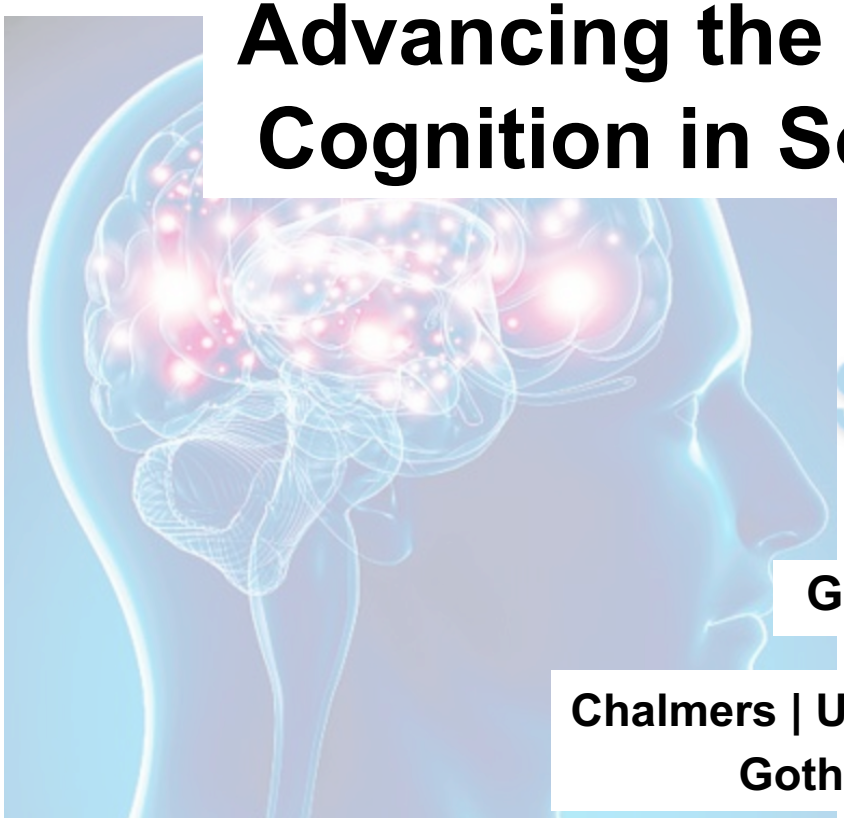
# Advancing the Research on Human Cognition in Software Engineering



**Gül Çalıklı**

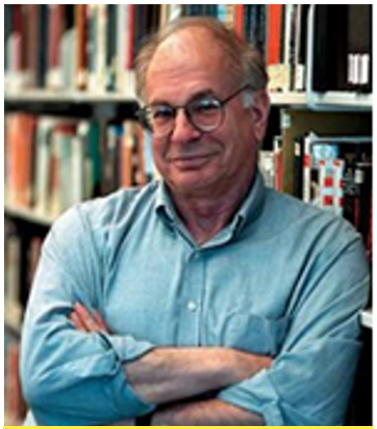
**Chalmers | University of Gothenburg**
**Gothenburg, Sweden**

# Cognitive Biases

- Cognitive biases are systematic deviations of human mind from optimal reasoning that produce errors in judgement.

# Cognitive Biases

Amos Tversky



Daniel Kahneman

## Judgment under Uncertainty: Heuristics and Biases

Biases in judgments reveal some heuristics of thinking under uncertainty.

Amos Tversky and Daniel Kahneman

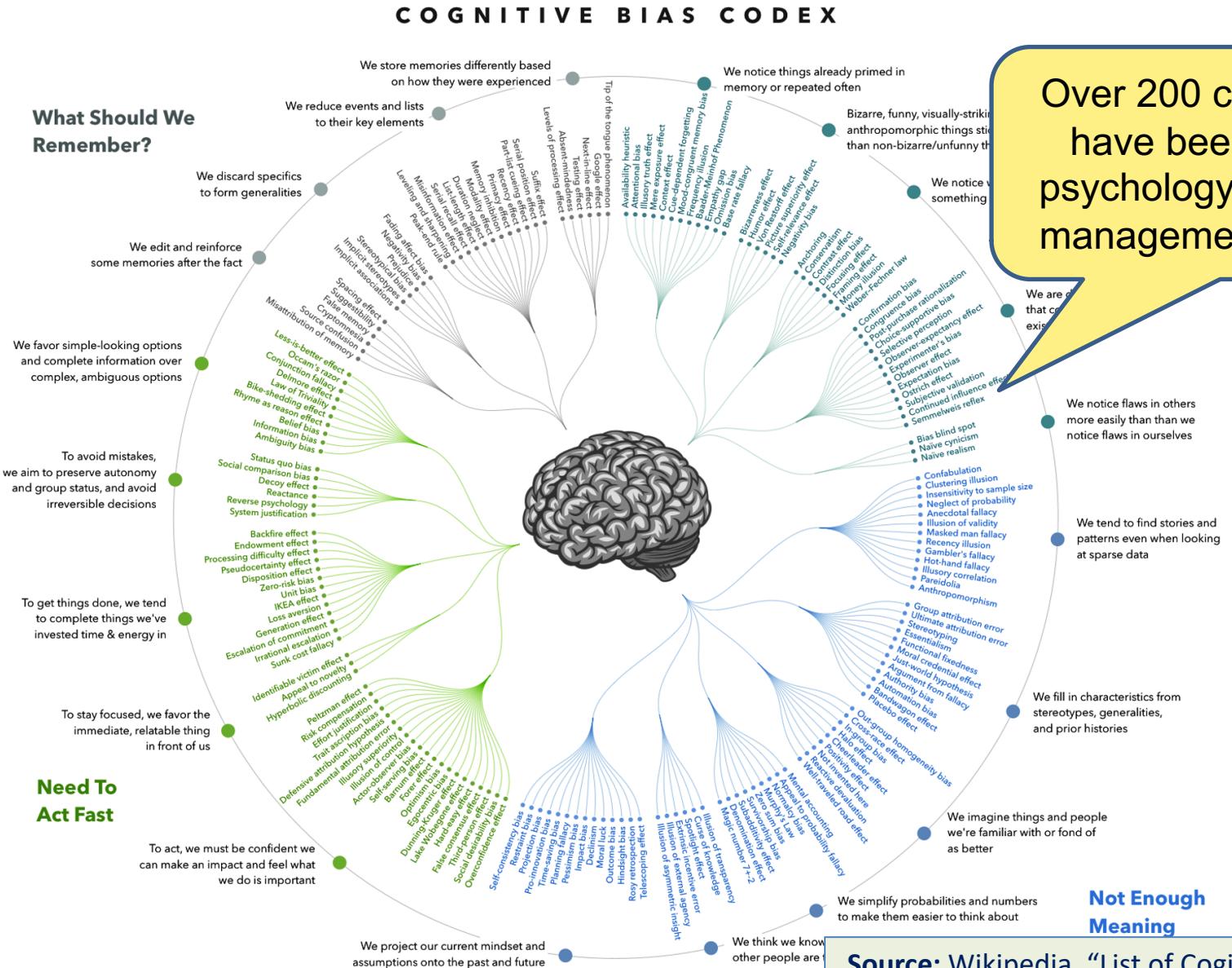
Many decisions are based on beliefs concerning the likelihood of uncertain events such as the outcome of an election, the guilt of a defendant, or the future value of the dollar. These beliefs are usually expressed in statements such as "I think that . . . ," "chances are . . . ," "it is unlikely that . . . ," and so forth. Occasionally, beliefs concerning uncertain events are expressed in numerical form as odds or subjective probabilities. What determines such beliefs? How do people assess the probability of an uncertain event or the mated when visibility is good because the objects are seen sharply. Thus, the reliance on clarity as an indication of distance leads to common biases. Such biases are also found in the intuitive judgment of probability. This article describes three heuristics that are employed to assess probabilities and to predict values. Biases to which these heuristics lead are enumerated, and the applied and theoretical implications of these observations are discussed.

occupation from a list of possibilities (for example, farmer, salesman, airline pilot, librarian, or physician)? How do people order these occupations from most to least likely? In the representativeness heuristic, the probability that Steve is a librarian, for example, is assessed by the degree to which he is representative of, or similar to, the stereotype of a librarian. Indeed, research with problems of this type has shown that people order the occupations by probability and by similarity in exactly the same way (*1*). This approach to the judgment of probability leads to serious errors, because similarity, or representativeness, is not influenced by several factors that should affect judgments of probability.

*Insensitivity to prior probability of outcomes.* One of the factors that have no effect on representativeness but should have a major effect on probability is the prior probability, or base-rate frequency, of the outcomes. In the case of Steve, for example, the fact that there are many more farmers than librarians in the population should enter into any reasonable estimate of the probability that Steve is a librarian rather than a farmer. Considerations of base-rate frequency, however, do not affect the similarity of Steve to the

* This article originally appeared in Science, vol. 185, 1974.

# Cognitive Biases

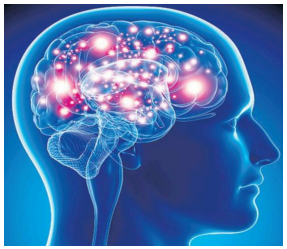



Over 200 cognitive biases have been identified in psychology, sociology and management research[1,2].

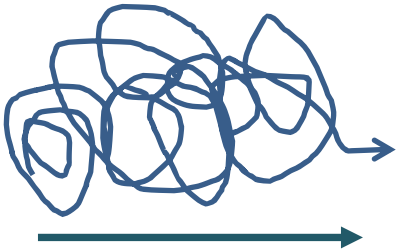
**Source:** Wikipedia, "List of Cogitive Biases", https://en.wikipedia.org/wiki/List_of_cognitive_biases

# Common Sources of Cognitive Biases

## Cognitive Limitations



lead to

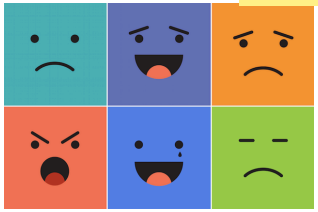


Limitations in information processing capacity (e.g., **memory**, **working memory**).

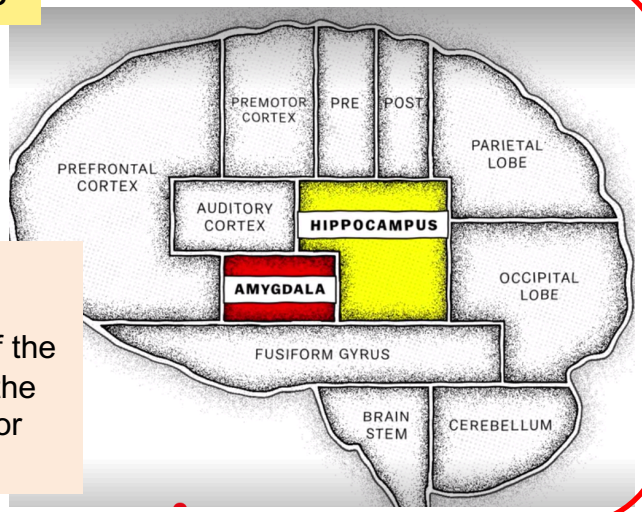
Mental short-cuts called ”heuristics”

## Individual Motivations

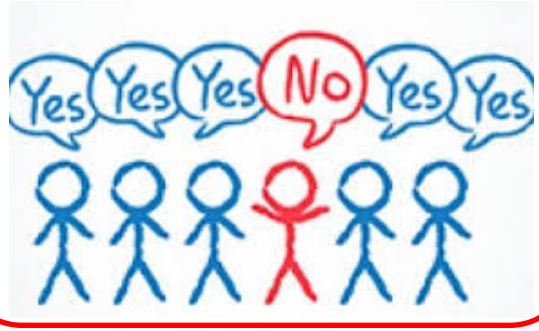

## Emotions



**Emotions facilitate memory:** When we have an emotional experience, emotional center of the brain “amygdala” up-regulates the hippocampus, which has a major role in memory.




## Social Pressure

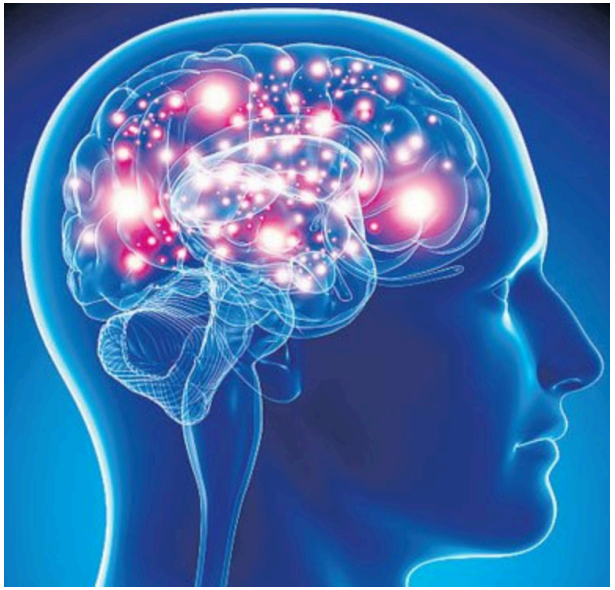


# Cognitive Biases in Software Engineering

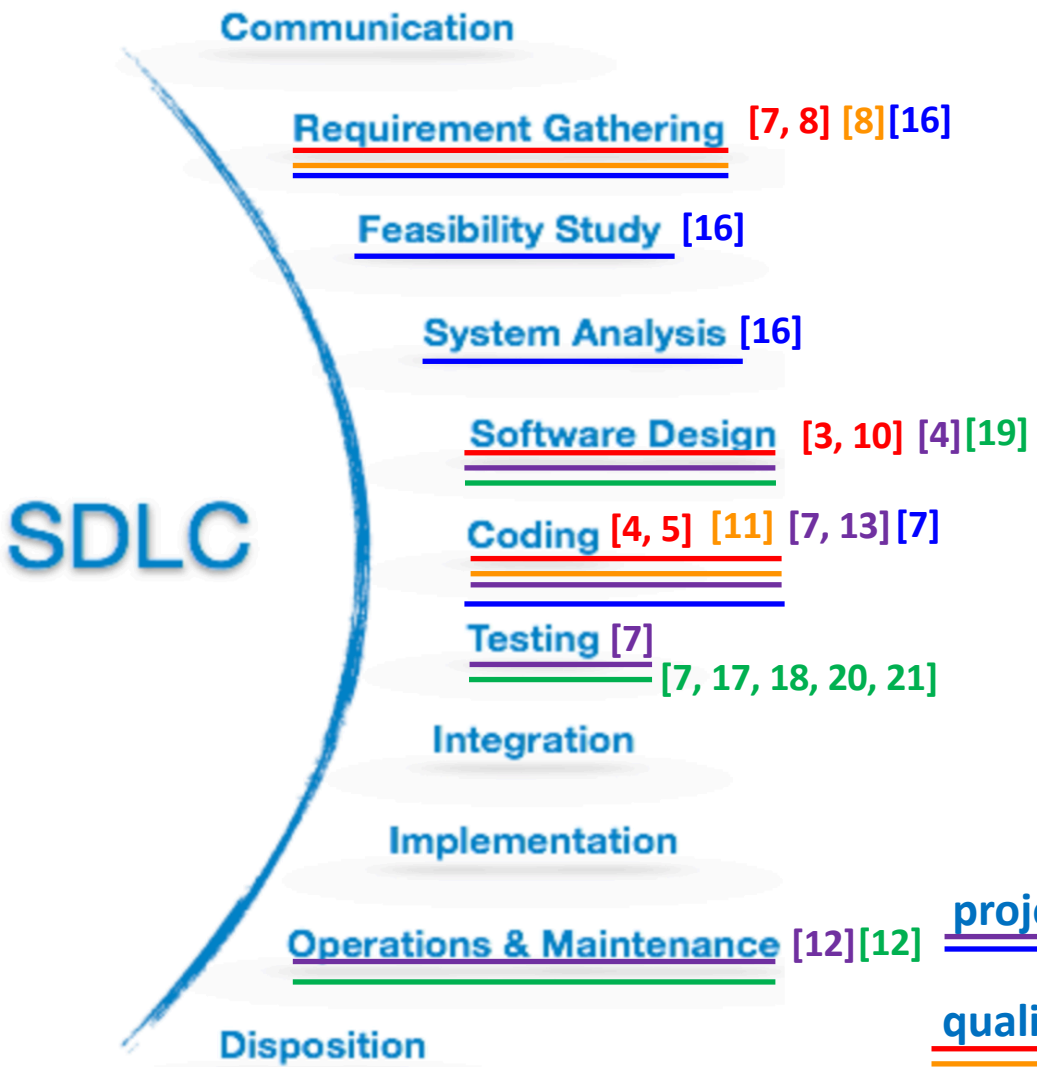
- Software is designed and developed by **people.**

# Cognitive Biases in Software Engineering


- There is involvement of **human judgement** in every stage of Software Development Life Cycle (SDLC).



# Some Examples for Studies in Cognitive Biases in SE




**Anchoring & Adjustment:** Making estimates by starting from an initial value that is adjusted to yield the final answer. Anchoring occurs when adjustments are insufficient.

**Overconfidence:**Tendency to be more confident in one's own abilities.

**Availability:**Tendency to deem what is easy to recall as being significant.

**Representativeness:** Assessing the probability of an event or item based on the similatitie of the features to the parent population.

**Confirmation Bias:** Favoring information that confirms our beliefs.

**project management** [9, 12, 14]

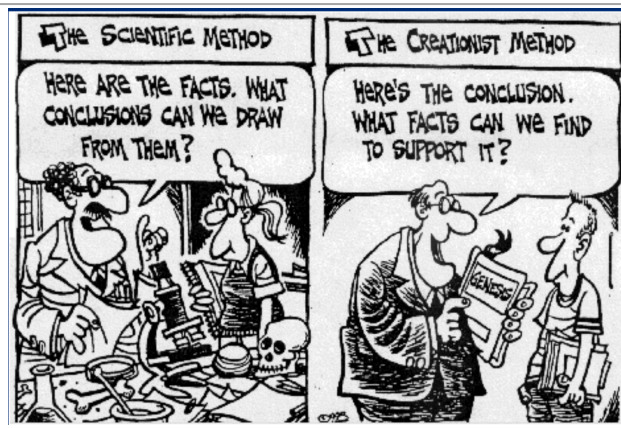
**quality management** [8, 11] [15] [6, 9]



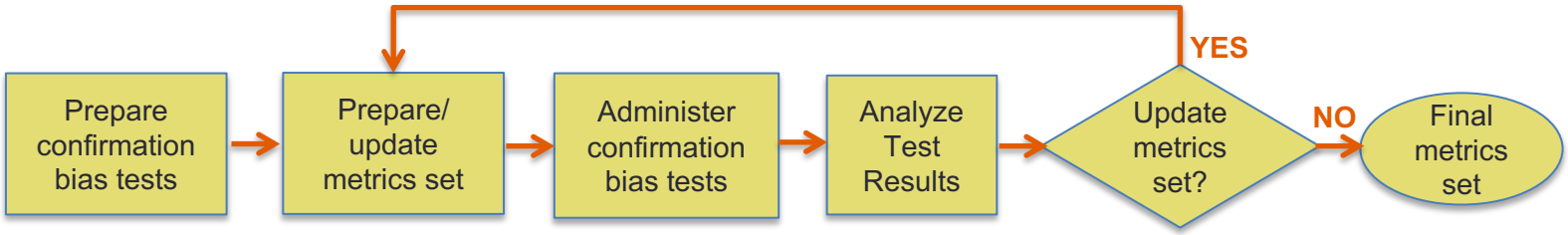
For further information → "R. Mohanani, I. Salman, B. Turhan, P. Rodríguez and P. Ralph, "Cognitive Biases in Software Engineering: A Systematic Mapping Study," in *IEEE Transactions on Software Engineering.*"

# My Previous Research: Confirmation Bias in SE

- **Confirmation Bias:** Tendency to find evidence that supports one's beliefs rather than finding evidence refuting them.
- Due to confirmation bias, developers perform unit tests to make their program work rather than to break their code*.




## Formation of a Metrics Set:




## Formation of the Metrics Repository:

- consists confirmation bias values of **199** software engineers:
  - **129** developers
  - **26** testers
  - **32** analysts
  - **12** project managers

IBM in Canada
The IBM Canada Software Labs

TURKCELL
VERIPARK
LOGO BUSINESS SOLUTIONS
FINANSBANK
INFINA
INNOVA future-ready solutions

*L. M. Leventhal, B. M. Teasley, D. S. Rohlman, and K. Instone, "Positive test bias in software testing among professionals: A review," in International Conference on Human—Computer Interaction. Springer, 1993, pp. 210—218.

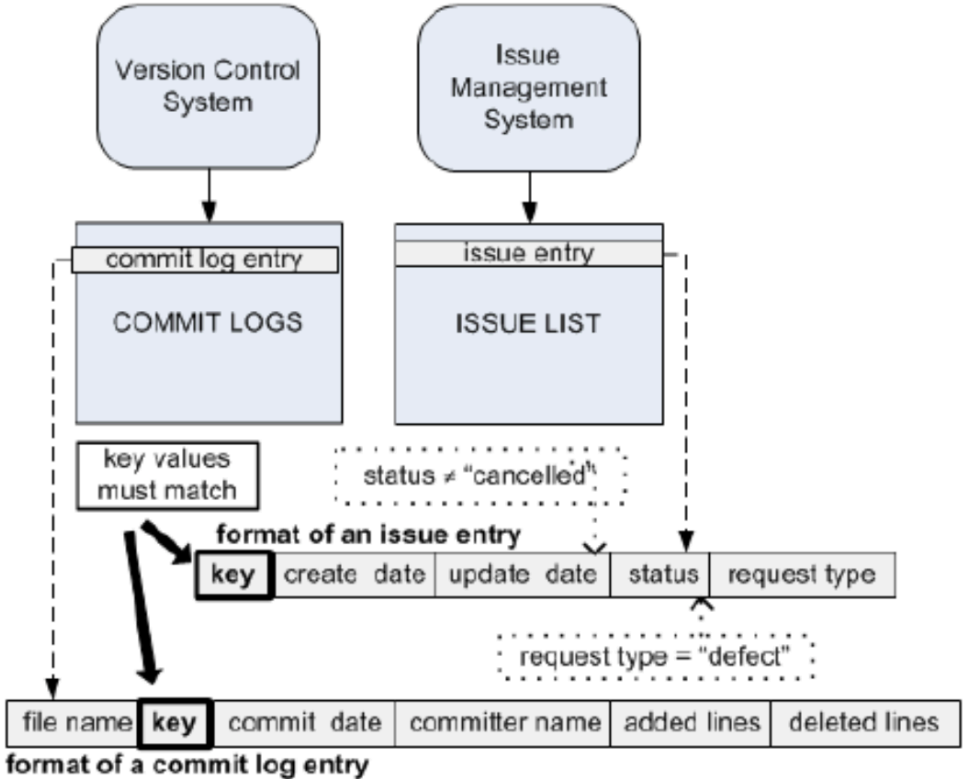
# My Previous Research: Confirmation Bias in SE

## Building Defect Prediction Models:

- **Algorithm:** Naive Bayes
- **Input data:** static code, churn, confirmation bias metrics
- **Pre-processing:** under-sampling
- 10x10 cross validation

**Results Summary:**

- Confirmation Bias is a single human aspect.
- Yet, using confirmation bias metrics we obtained comparable performance results
- Therefore, we should further investigate human aspects...



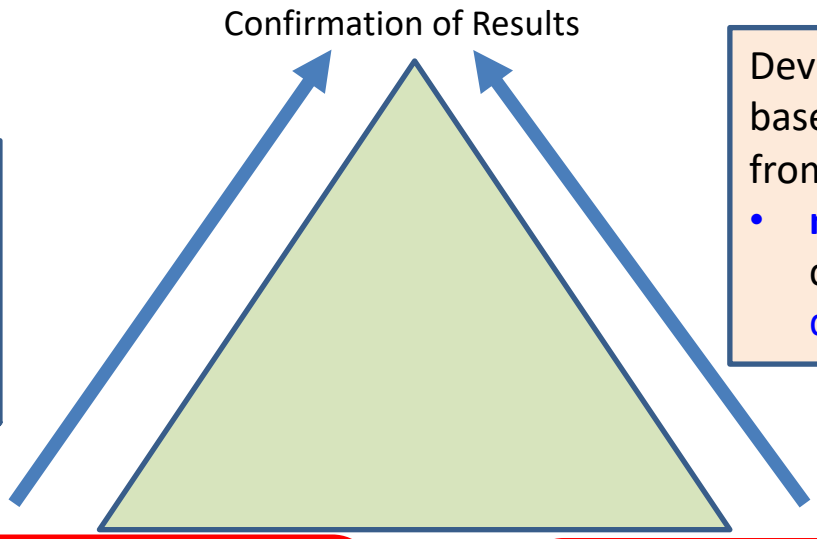
# Cognitive Biases in SE: Research Gap

Research Gap #1:

Is the observed phenomenon manifestation of the claimed cognitive bias?

# Cognitive Biases in SE: Research Gap

**Proposed Solution:** Triangulation




**Mediating process** to trigger cognitive bias X must be similar in nature both in **SE experiment** and in the **measurement setup**.

Devise a **measurement set-up** based on a psychology experiment from the literature, such that:

- **mediating process** to trigger cognitive bias X is manipulated directly.

**SE Experiment:** Treatment aims to trigger cognitive bias X.

**Measurement Set-up** to measure participants' proneness to cognitive bias X.

# Application of Proposed Solution

accepted at ICSE'20

## 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**

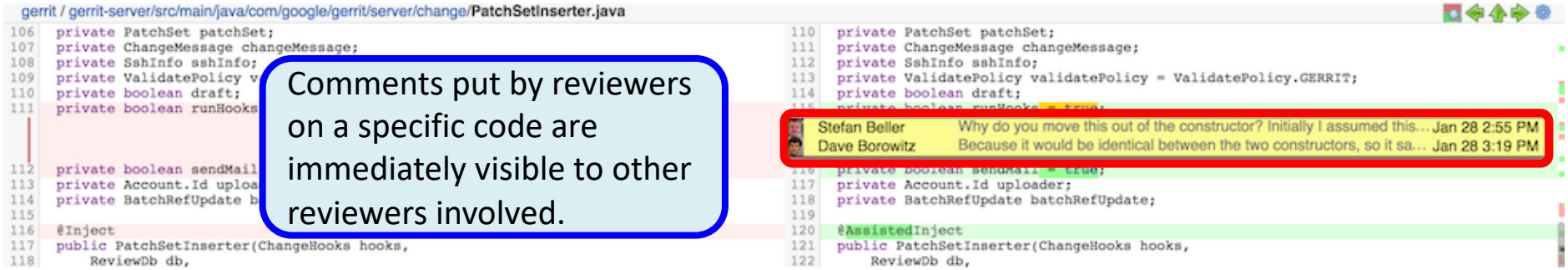
In contemporary code review, the comments put by reviewers on a specific code change are immediately visible to the other reviewers involved. Could this visibility prime new reviewers' attention (due to the human's proneness to availability bias), thus biasing the code review outcome? In this study, 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

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

# Contemporary Code Review

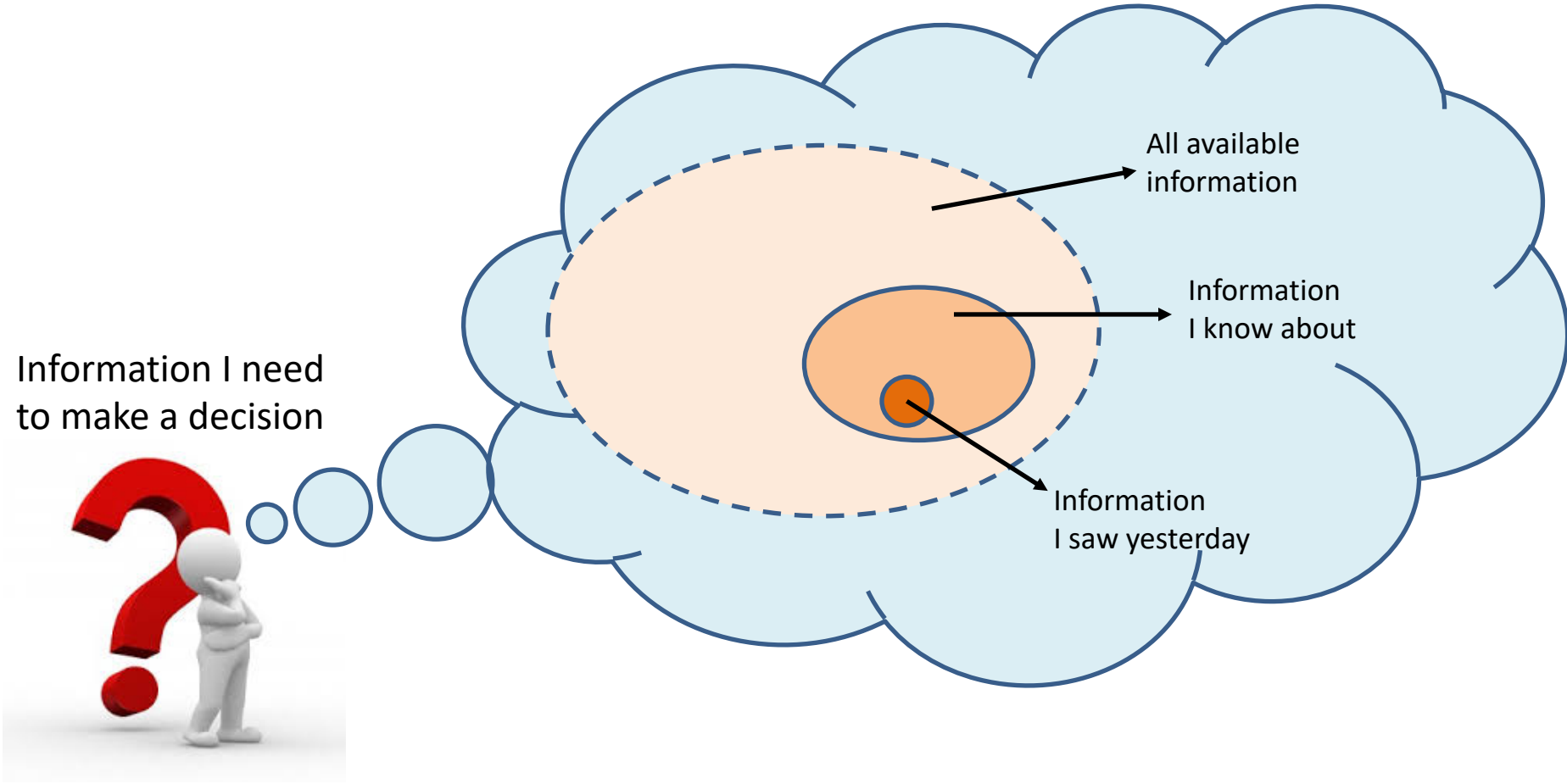



Could this visibility prime reviewers' attention (due to proneness to availability bias) and thus bias review outcome?

What is availability bias?

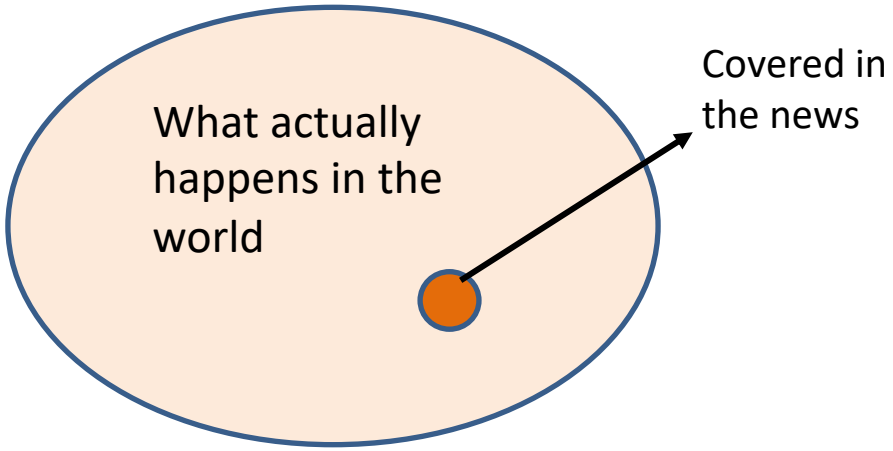
# Availability Bias

What comes to mind quickly (i.e., easy to recall) is deemed significant – sometimes incorrectly.



# Some Examples of Availability Bias

- A salient event that attracts one's attention (e.g., divorces of celebrities).
- A dramatic event one has witnessed or seen on news (e.g., a plane crash on the news, seeing a burning car on the side of the road).
- Personal experiences (e.g., a judicial error that affects you undermines your faith in justice system.)
- Recenty being exposed to some phenomena (e.g., watching a spy movie and then seeing conspiracies everywhere).

# Availability Bias in Contemporary Code Review

gerrit / gerrit-server/src/main/java/com/google/gerrit/server/change/**PatchSetInserter.java**

```
106  private PatchSet patchSet;
107  private ChangeMessage changeMessage;
108  private SshInfo sshInfo;
109  private ValidatePolicy validatePolicy = ValidatePolicy.GERRIT;
110  private boolean draft;
111  private boolean runHooks;
112  private boolean sendMail;
113  private Account.Id uploader;
114  private BatchRefUpdate batchRefUpdate;
115
116  @Inject
117  public PatchSetInserter(ChangeHooks hooks,
118      ReviewDb db,
```

```
110  private PatchSet patchSet;
111  private ChangeMessage changeMessage;
112  private SshInfo sshInfo;
113  private ValidatePolicy validatePolicy = ValidatePolicy.GERRIT;
114  private boolean draft;
116  private boolean sendMail = true;
117  private Account.Id uploader;
118  private BatchRefUpdate batchRefUpdate;
119
120  @AssistedInject
121  public PatchSetInserter(ChangeHooks hooks,
122      ReviewDb db,
```

Stefan Beller — Why do you move this out of the constructor? Initially I assumed this... Jan 28 2:55 PM
Dave Borowitz — Because it would be identical between the two constructors, so it sa... Jan 28 3:19 PM

Could visibility of previously made comments prime reviewers' attention towards (a) specific bug type(s) and thus affect review outcome?

Could such priming result in overlooking of other bug types?

Are the current code review settings robust to such priming despite developers' potential proneness to availability bias?

Some questions we asked ourselves during the initiation of this study…

How prone are developers to availability bias?

# Availability Bias in Contemporary Code Review

gerrit / gerrit-server/src/main/java/com/google/gerrit/server/change/**PatchSetInserter.java**

```
106  private PatchSet patchSet;
107  private ChangeMessage changeMessage;
108  private SshInfo sshInfo;
109  private ValidatePolicy validatePolicy = ValidatePolicy.GERRIT;
110  private boolean draft;
111  private boolean runHooks;
112  private boolean sendMail;
113  private Account.Id uploader;
114  private BatchRefUpdate batchRefUpdate;
115
116  @Inject
117  public PatchSetInserter(ChangeHooks hooks,
118      ReviewDb db,
```

```
110  private PatchSet patchSet;
111  private ChangeMessage changeMessage;
112  private SshInfo sshInfo;
113  private ValidatePolicy validatePolicy = ValidatePolicy.GERRIT;
114  private boolean draft;
115  private boolean runHooks = true;
116  private boolean sendMail = true;
117  private Account.Id uploader;
118  private BatchRefUpdate batchRefUpdate;
119
120  @AssistedInject
121  public PatchSetInserter(ChangeHooks hooks,
122      ReviewDb db,
```

Stefan Beller: Why do you move this out of the constructor? Initially I assumed this... Jan 28 2:55 PM

Dave Borowitz: Because it would be identical between the two constructors, so it sa... Jan 28 3:19 PM

Could visibility of previously made comments prime reviewers' attention towards (a) specific bug type(s) and thus affect review outcome?

Could such priming result in overlooking of other bug types?

Is the current code review settings robust to such priming despite developers' potential proneness to availability bias?

How prone are developers to availability bias?

Some questions we asked ourselves during the initiation of this study…

Taking one step back and we asked

How would this priming induced by previously made comments about bugs afffect code review outcome?

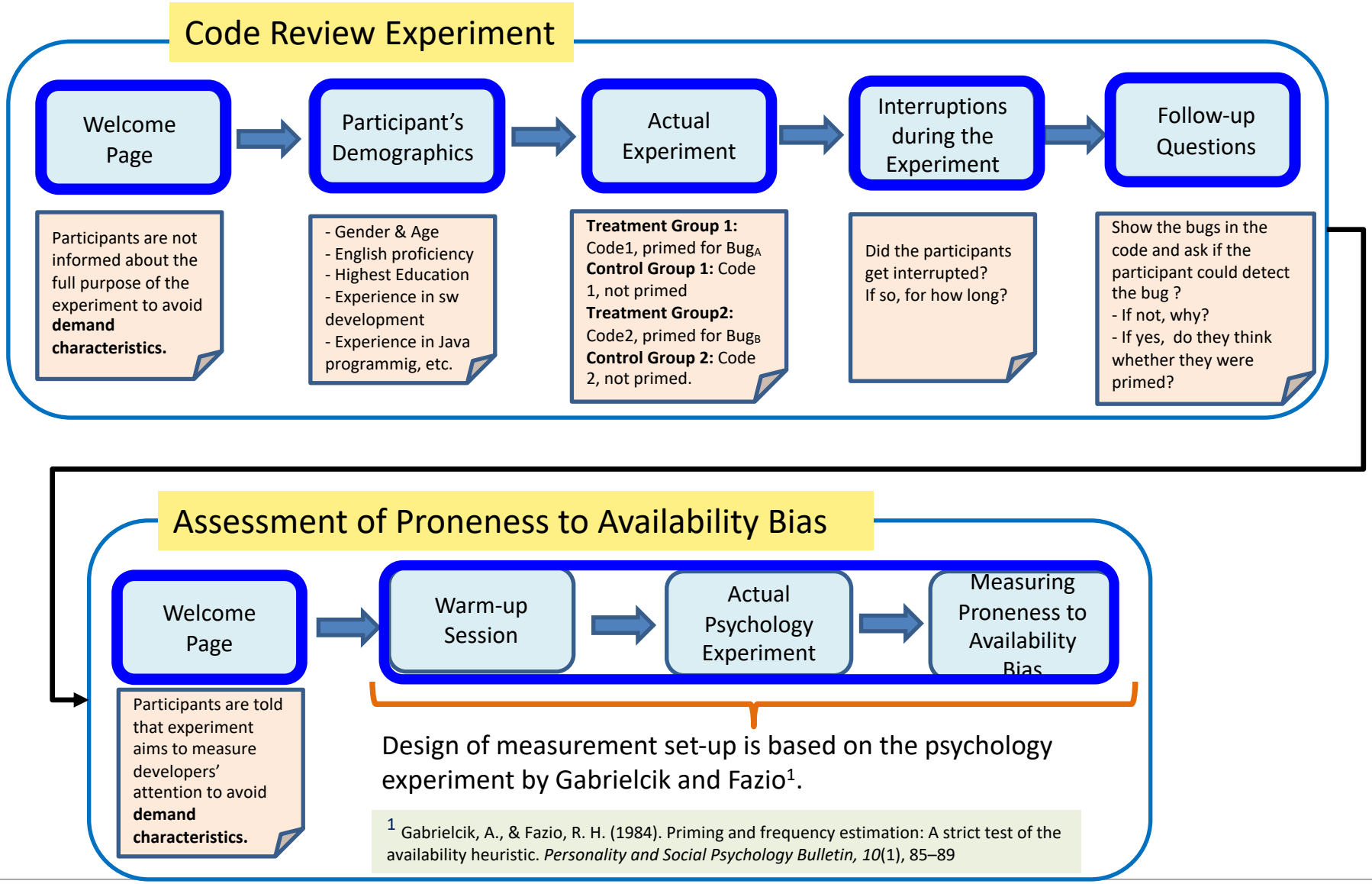
Does the priming effect change with respect to the bug type (e.g., bugs normally considered vs. bugs normally not considered)?

# Research Questions

**RQ1:** What is the effect of priming the reviewer with a bug that is **not** normally considered?

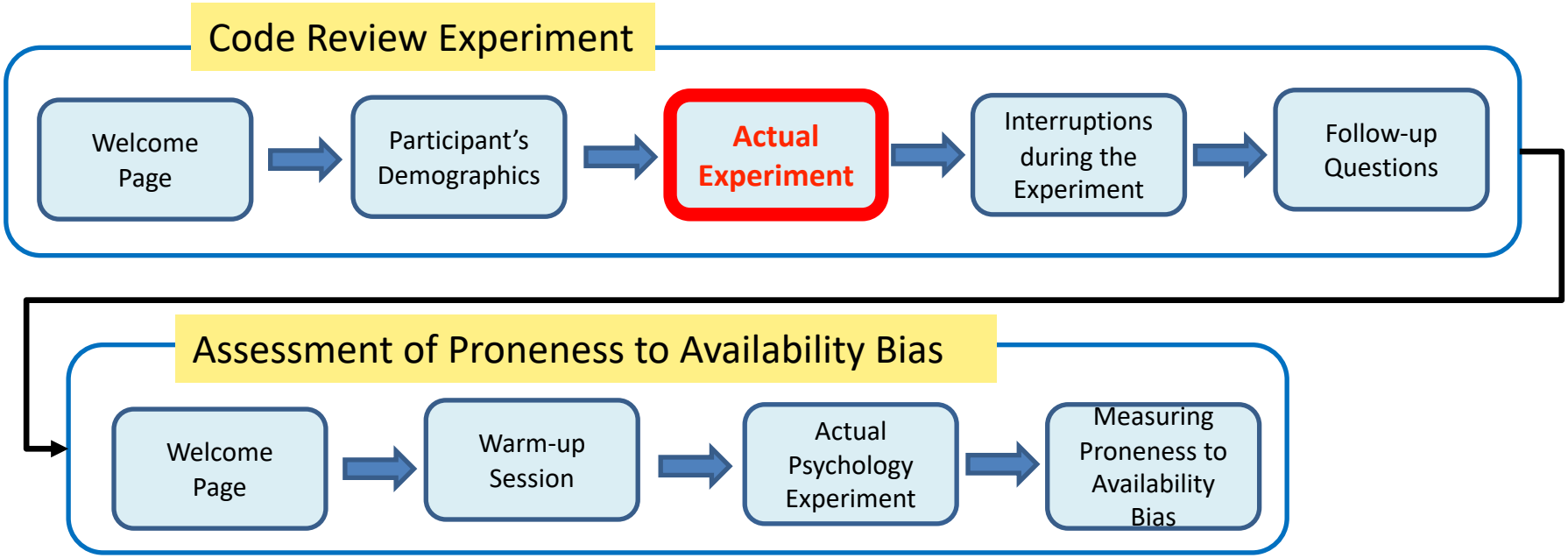
**RQ2:** What is the effect of priming the reviewer with a bug that is normally considered?

# Experimental Design




[1] Gabrielcik, A., & Fazio, R. H. (1984). Priming and frequency estimation: A strict test of the availability heuristic. *Personality and Social Psychology Bulletin, 10*(1), 85–89

# Experimental Design



# Actual Code Review Experiment

**RQ1:** What is the effect of priming the reviewer with a bug that is **not** normally considered (i.e., $BUG_A$)?

**Treatment Group 1**

Prime with $BUG_A$ (i.e., a reviewer comment for one of the bugs of type $BUG_A$ exists on code change)

2 bugs of type $BUG_A$ and 1 bug of type $BUG_B$ are injected into the code change.

- **$BUG_A$:** Bugs that's are **NOT** normally considered (e.g., NullPointerException*)
- **$BUG_B$:** Bugs that's are normally considered (e.g., Corner case bugs*)

**Control Group 1**

No reviewer comments (i.e., no priming).

# Actual Code Review Experiment

**RQ2:** What is the effect of priming the reviewer with a bug that is normally considered (i.e., $BUG_B$)?

**Treatment Group 2**

Prime with $BUG_B$ (i.e., a reviewer comment for one of the bugs of type $BUG_B$ exists on code change)

2 bugs of type $BUG_B$ and 1 bug of type $BUG_A$ are injected into the code change.

- **$BUG_A$:** Bugs that's are **NOT** normally considered (e.g., NullPointerException*)
- **$BUG_B$:** Bugs that's are normally considered (e.g., Corner case bugs*)

**Control Group 2**

No reviewer comments (i.e., no priming).

# Actual Code Review Experiment: A Screenshot

## Instructions

We are now going to show you the code changes to review. The old version of the code is on the left, the new version is on the right.

For the scientific validity of this experiment, it is vital that the review task is taken **very seriously**.

- Like in real life, you should **find as many defects as possible** and you should **spend as little time as possible** on the review.
- Unlike in real life, we are **not interested in maintainability or design issues**, but only in correctness issues ("bugs").

For example, a remark like the following is beyond the goal of the review: "Create a new class which is implemented by runnable interface that we can access multiple times." Instead, what we are interested in are the defects that make the code not work as intended under all circumstances.

Please assume that the code compiles and that the tests pass.

**You will see that a previous reviewer already put a comment in line 23**. You are now asked to continue with your review.

To add a review remark, click on the corresponding line number. To delete a review mark, click on it again and delete the remark's text.

src/main/java/org/pack/ExerciseSumArray.java

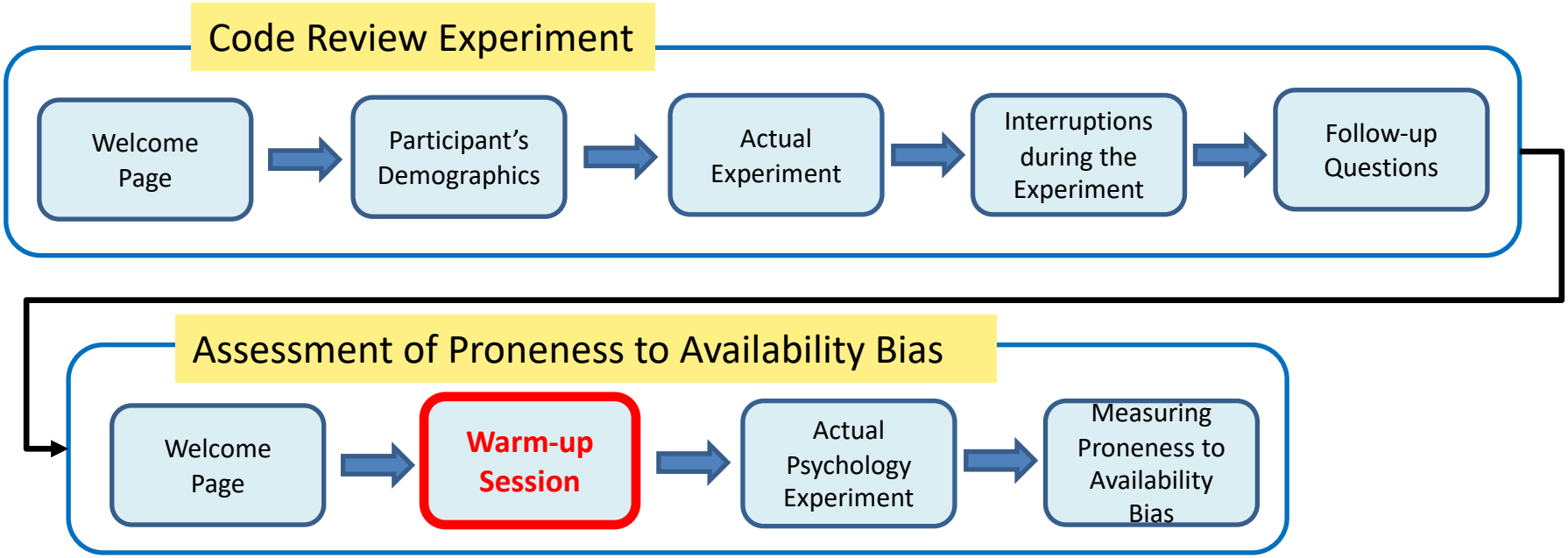
```
1  public class ExerciseSumArray {
2      /*
3      Given 2 Lists representing numbers (e.g., [3,4] = 34, [9,8] = 98),
4      calculate the sum of 2 Lists, and return the result in an List.
5      For example:
6      [1, 0, 0] + [4,0] = [1,4,0]
7      [6,7] + [0] = [6,7]
8       */
9  }
10
```

Sreenshot of the online experiment given to Treatment Group 2

src/main/java/org/pack/ExerciseSumArray.java

```
1  public class ExerciseSumArray {
2      /*
3      Given 2 Lists representing numbers (e.g., [3,4] = 34, [9,8] = 98),
4      calculate the sum of 2 Lists, and return the result in an List.
5      For example:
6      [1, 0, 0] + [4,0] = [1,4,0]
7      [6,7] + [0] = [6,7]
8       */
9      public ArrayList<Integer> getSum(List<Integer> firstNumber, List<Integer> secondNumber ){
10         ArrayList<Integer> result = new ArrayList<Integer>();
11
12         int carry = 0;
13         Collections.reverse(firstNumber);
14         Collections.reverse(secondNumber);
15
16         for (int i = 0; (i < Math.max(firstNumber.size(), secondNumber.size())); i ++){
17             Integer firstValue = i < firstNumber.size() ? firstNumber.get(i) : null;
18             Integer secondValue = i < secondNumber.size() ? secondNumber.get(i) : null;
19
20             int res = firstValue + secondValue + carry;
21
22             carry = 0;
23             if (res > 10){
(!) Pat Smith: This is a bug related to a corner cases. The check should be >=, otherwise it fails in assigning the carry (e.g. 29 + 1).
24                 carry = 1;
25                 res = res % 10;
26             }
27             result.add(res);
28         }
29
30         if (carry >= 0)
31             result.add(carry);
32
33         Collections.reverse(result);
34         return result;
35     }
36 }
```

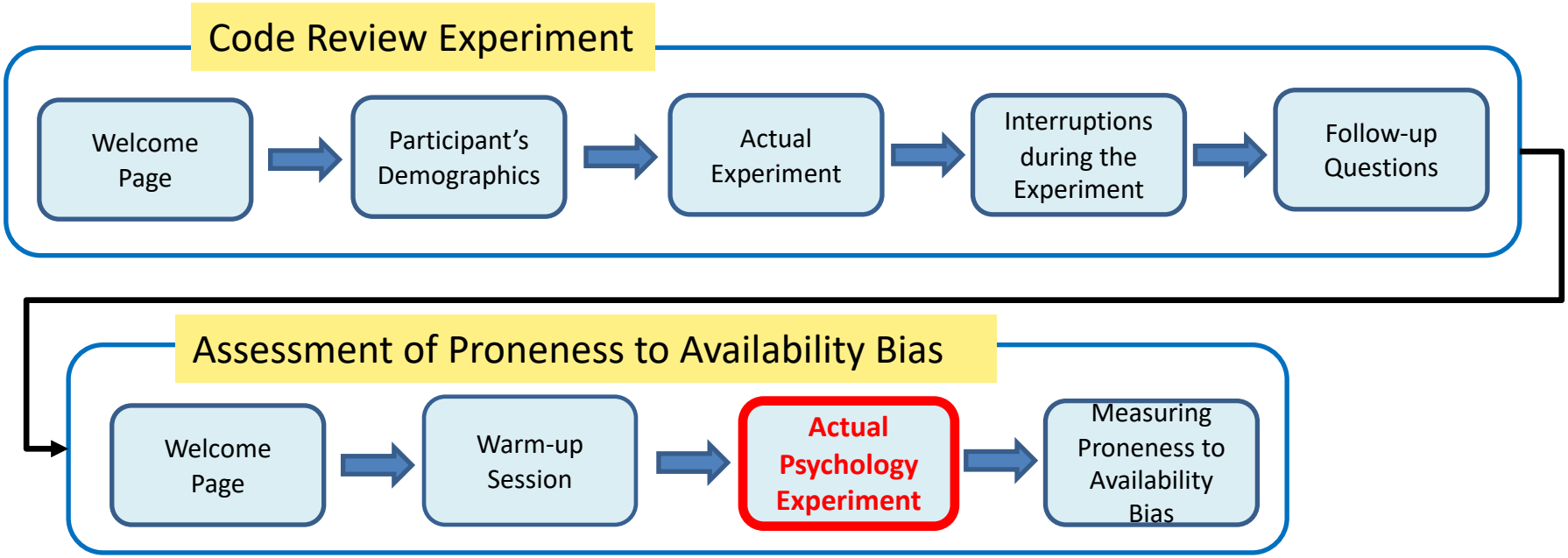
# Experimental Design



# Psychology Experiment: Warm-up Session

- Participants are asked to focus on a series of 20 words flashing on the screen.
- Words are randomly selected from the English Dictionary.
- **None of the words contain letter "T"**.
- Each word flashes on the screen for 300 milliseconds.
- At the end of the session participants are asked to write 3 words they have seen.

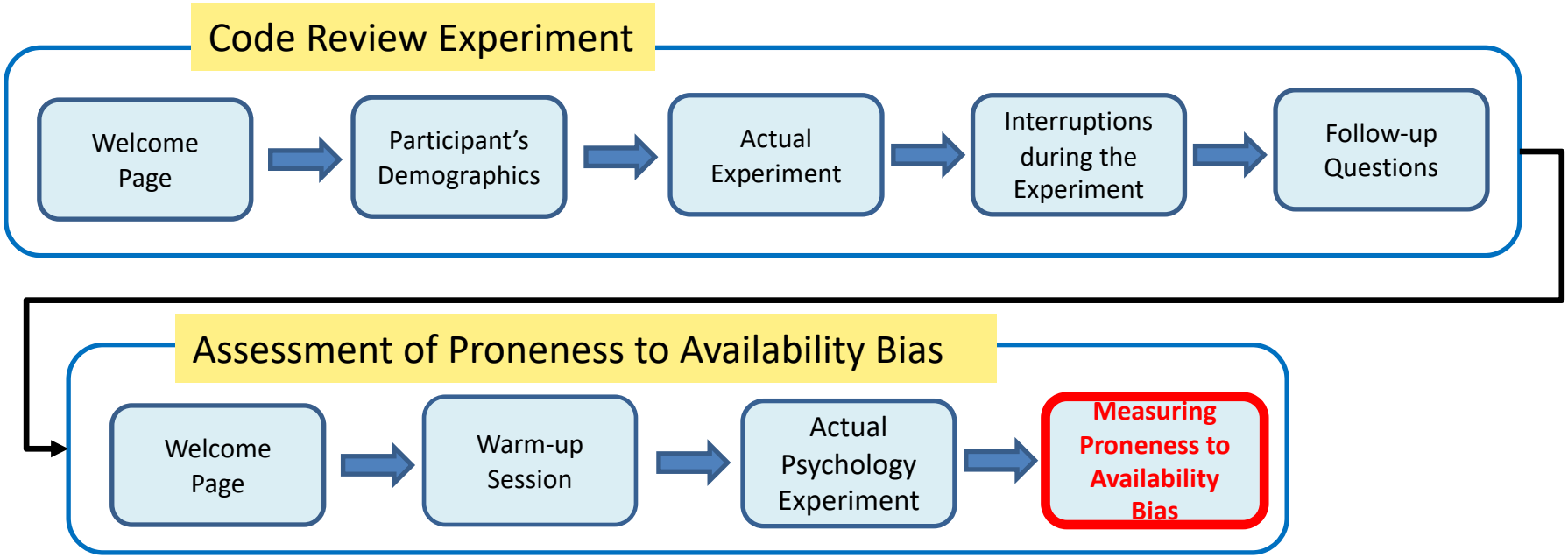
# Experimental Design



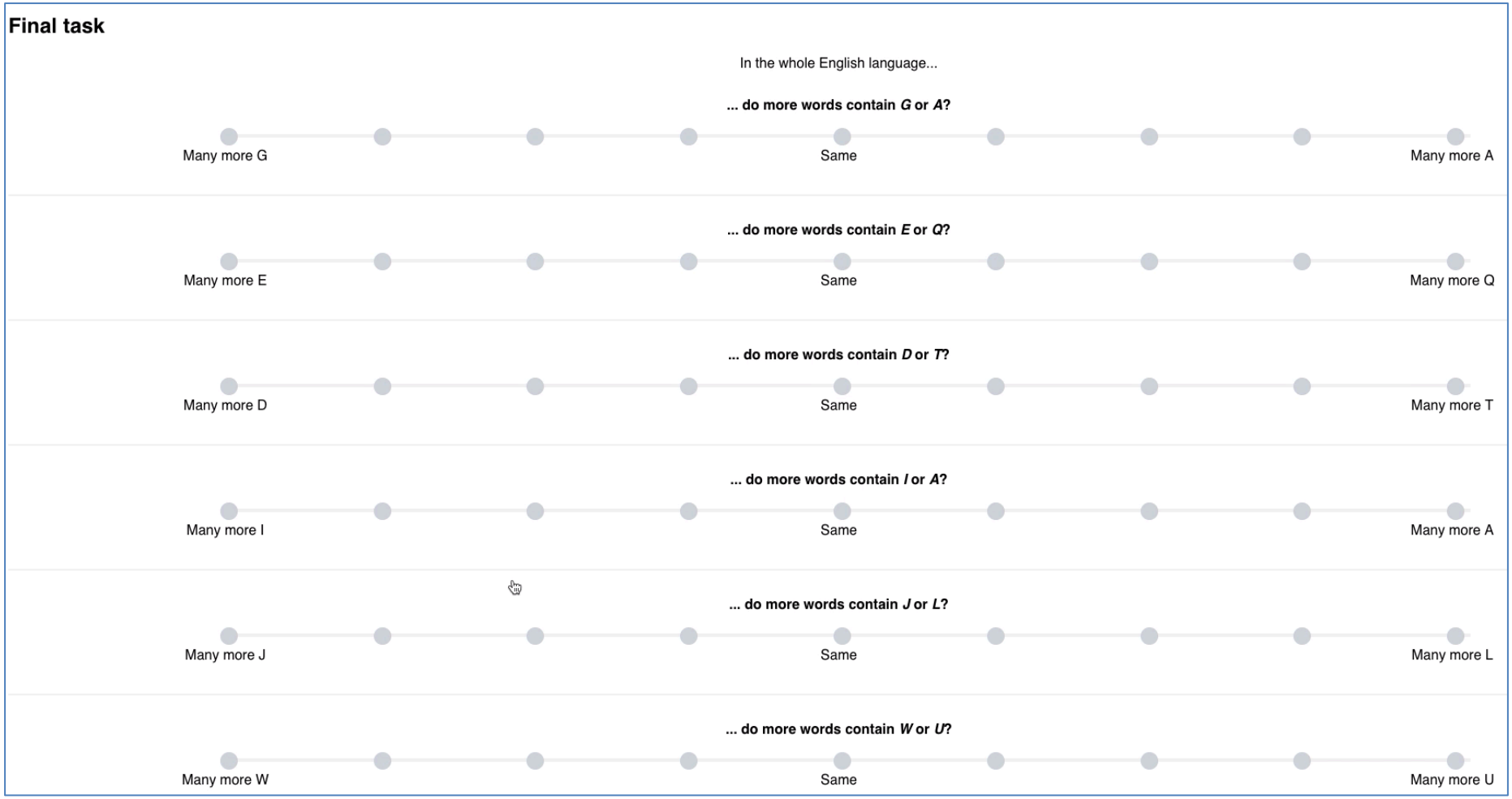
# Actual Psychology Experiment

- We show 2 series of 20 words, each.
- Words are randomly selected from the English Dictionary.
- **Each word contains at least one letter “T”.**
- Each word flashes on the screen for 150 milliseconds.
- At the end each series, participants are asked to write 3 words they have seen.

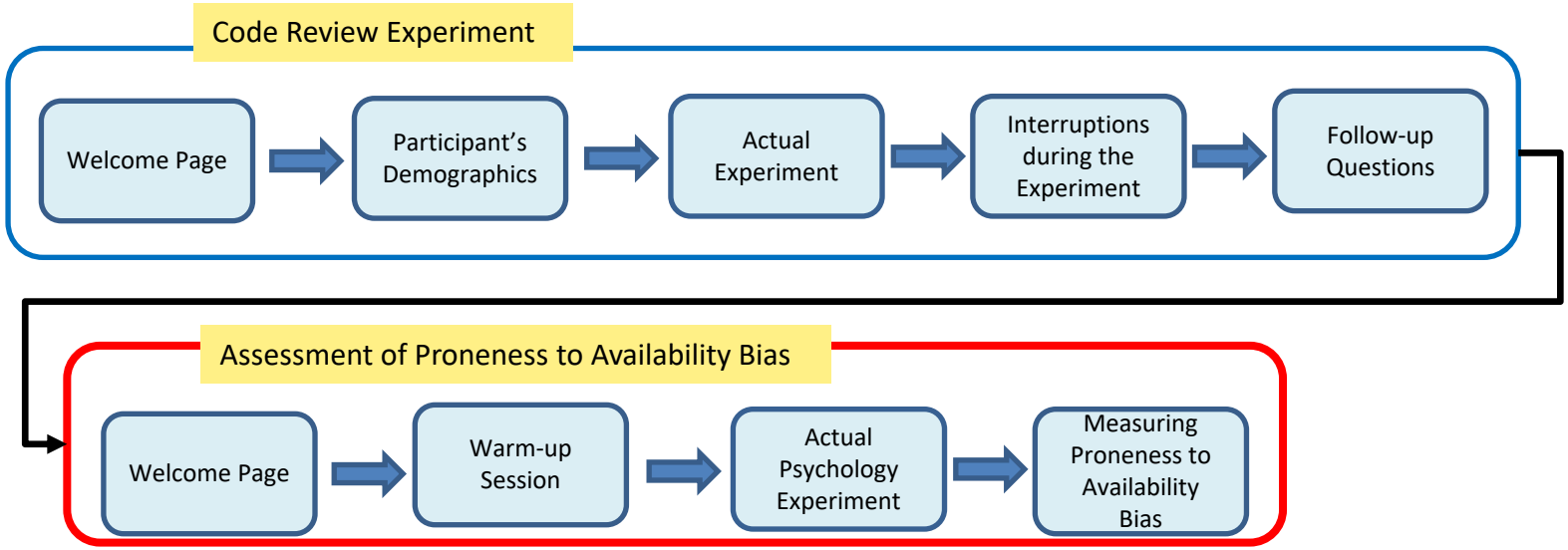
# Experimental Design

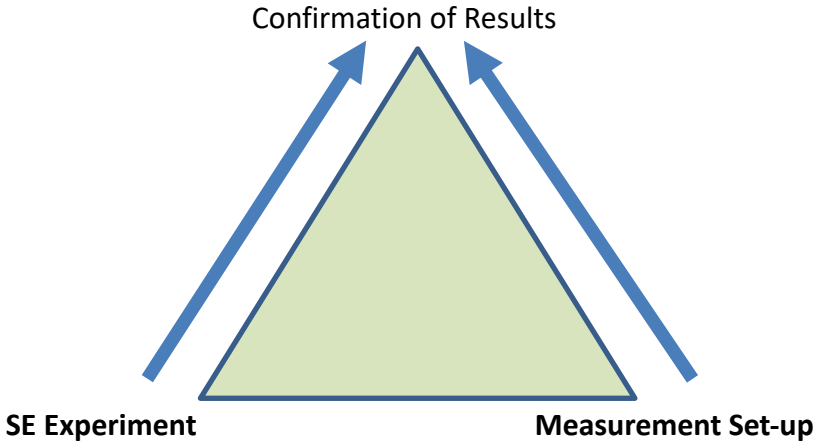


# Measuring Proneness to Availability Bias



# How Triangulation is employed in this study…

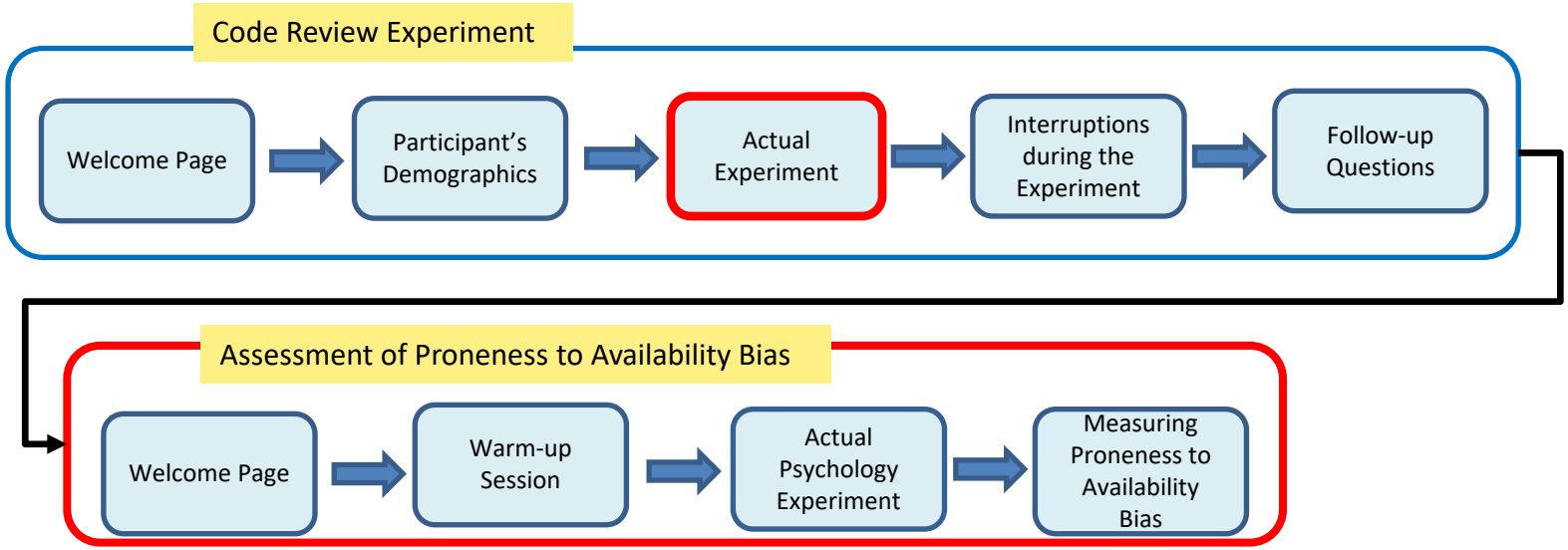


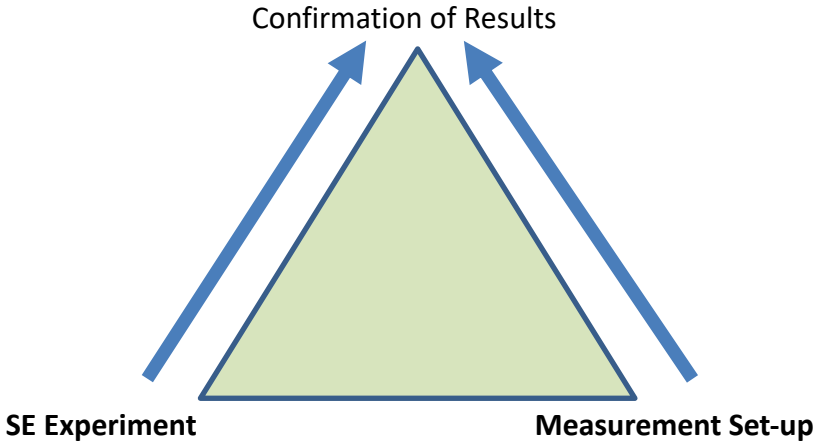




We devised a **measurement set-up** based on a psychology experiment from the literature, such that:

- **mediating process (i.e., memory priming)** to trigger availability bias is manipulated directly.

# How Triangulation is employed in this study…




**Mediating process** to trigger availability bias is **memory priming** both in **code review experiment** and in the **setup** for the **assessment of proneness to availability bias**.

# Findings

| Primed Bug (NPE) | Treatment Group1 | Control Group1 | Total |
|---|---|---|---|
| found | 13 | 2 | 15 |
| not found | 8 | 15 | 23 |
| | Odds Ratio: | 12.19 (2.19, 67.94) | |
| | | | **p < 0.001** |

| Not Primed Bug (CC) | Treatment Group1 | Control Group1 | Total |
|---|---|---|---|
| | | | 28 |
| | | | 10 |
| | | | |
| | | | **p < 0.275** |

***Finding 1:***
*Reviewers primed on a bug that is **not** normally considered (e.g., NPE) are more likely to find other occurrences of this type of bugs.*

*However, this does not prevent them from finding also other types of bugs.*

Odds ratio for capturing the primed and not primed bug:

**Primed Bug:** NullPointerException (NPE)
**Not primed Bug:** Corner Case (CC)

| | | | | Not Primed Bug | | |
|---|---|---|---|---|---|---|
| | | | | | S.E. | Sig. |
| Intercept | 0.704 | 4.734 | | -0.893 | | |
| IsPrimed | 3.627 | 1.320 | ** | -1.199 | 1.073 | |
| TotalDuration | 0.001 | 0.002 | | 0.003 | 0.001 | * |
| ProfDevExp | 0.813 | 0.557 | | -0.503 | 0.554 | |
| ProgramPractice | -0.096 | 0.828 | | -0.243 | 0.736 | * |
| .... | | | | | | |
| Interruptions | -1.752 | 0.758 | * | -0.715 | 0.444 | |

Regression for the primed and not primed bug:

*Significance codes:*
*** p < 0.001,
** p < 0.01
* p < 0.1

# Findings

| Primed Bug (CC) | Treatment Group2 | Control Group2 | Total |
|---|---|---|---|
| found | 10 | 8 | 18 |
| not found | 12 | 17 | 29 |
| | Odds Ratio: | 1.77 (0.54, 5.81) | |
| | | | **p < 0.344** |

| Not Primed Bug (NPE) | Treatment Group2 | Control Group2 | Total |
|---|---|---|---|
| | | 13 | 29 |
| | | 9 | 18 |
| | | 1) | |
| | | | **p < 0.73** |

***Finding 2:***
*Reviewers primed on a bug that is normally considered (e.g., CC) perceive an influence, but are as likely as the other to find bugs of this type.*

*Furthermore, primed participants did not capture fewer bugs of other type.*

Odds ratio for capturing the primed and not primed bug:

**Primed Bug:** Corner Case (CC)
**Not primed Bug:** NullPointerException (NPE)

| | | | | Not Primed Bug | | |
|---|---|---|---|---|---|---|
| | | | | ate | S.E. | Sig. |
| Intercept | 1.051 | 4.734 | | 3.037e-01 | 2.568 | |
| IsPrimed | 0.926 | 0.722 | * | -1.670e-01 | 7.74e-01 | |
| TotalDuration | 0.001 | 0.001 | . | 9.561e-05 | 3.721e-01 | |
| ProfDevExp | 0.813 | 0.557 | | -1.061 | 7.353e-01 | |
| ProgramPractice | 1.153 | 0.378 | | 1.211 | 4.683e-01 | ** |
| .... | | | | | | |
| Interruptions | -0.175 | 0.322 | | -0.715 | 0.444 | |

Regression for the primed and not primed bug:

*Significance codes:*
**** p < 0.001,*
*** p < 0.01*
** p < 0.05*
*. P < 0.1*

# Conclusions

- **GOAL:** To test the robustness of peer code review against reviewers' potential proneness to availability bias.
- **Methodology:** Online experiment conducted with 85 participants.
- **Psychology Experiment Results:** Majority of the participants (~%70) are prone to availability bias (median = 3.8, max = 4).

# Conclusions

- **Code Review Experiment Results** show that when reviewers are primed for:
  - a bug that is normally considered:
    - this does **not** affect their performance in finding bugs.
  - a bug that is normally **NOT** considered:
    - this increases their likelihood of finding bugs of similar type,
    - without affecting their performance in finding other types of bugs.

Existing comments act as *(positive)* ***reminders*** rather than *(negative) primers.*

# Cognitive Biases in SE: Research Gap

Research Gap #2:

Mediating processes that manifest cognitive biases (e.g., What happens in memory, working memory, etc.?)

Why is understanding mediating processes important?

# Cognitive Biases in SE: Research Gap

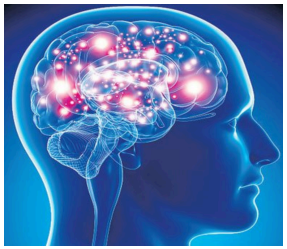
Research Gap #2:

Mediating processes that manifest cognitive biases (e.g., What happens in memory, working memory, etc.?)

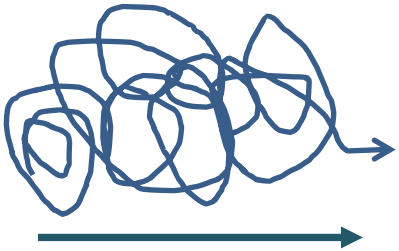

It can help development of tools/techniques for de-biasing.

# Back to Common Sources of Cognitive Biases

## Cognitive Limitations



lead to



Limitations in information processing capacity (e.g., **memory**, **working memory**).

Mental short-cuts called ”heuristics”

## Individual Motivations




## Emotions






**Emotions facilitate memory:** When we have an emotional experience, emotional center of the brain “amygdala” up-regulates the hippocampus, which has a major role in memory.

## Social Pressure

# Bounded Rationality

**Bounded rationality** is the idea that rationality is limited, when individuals make decisions, by the:

- tractability of the decision problem,
- cognitive limitations of the mind (e.g., memory, working memory), and
- time available to make the decision.

Herbert A. Simon

**Cognitive biases** are a "by-product" of human processing limitations, resulting from a lack of appropriate mental mechanisms or simply from a limited capacity for information processing (e.g., memory, working memory).

# Towards Understanding Working Memory…

ESEC/FSE 2019

## Effects of Explicit Feature Traceability on Program Comprehension


Jacob Krüger
Otto-von-Guericke University
Magdeburg, Germany
jkrueger@ovgu.de

Gül Çalıklı
Chalmers | University of Gothenburg
Gothenburg, Sweden
calikli@chalmers.se

Thorsten Berger
Chalmers | University of Gothenburg
Gothenburg, Sweden
bergert@chalmers.se

Thomas Leich
Harz University & METOP GmbH
Wernigerode & Magdeburg, Germany
tleich@hs-harz.de

Gunter Saake
Otto-von-Guericke University
Magdeburg, Germany
saake@ovgu.de



### ABSTRACT

Developers spend a substantial amount of their time with program comprehension. To improve their comprehension and refresh their memory, developers need to communicate with other developers, read the documentation, and analyze the source code. Many studies show that developers focus primarily on the source code and that small improvements can have a strong impact. As such, it is crucial to bring the code itself into a more comprehensible form. A particular technique for this purpose are explicit feature traces to easily identify a program's functionalities. To improve our empirical understanding about the effect of feature traces, we report an online experiment with 49 professional software developers. We studied the impact of explicit feature traces, namely annotations and decomposition, on program comprehension and compared them to the same code without traces. Besides this experiment, we also asked our participants about their opinions in order to combine quantitative and qualitative data. Our results indicate that, as opposed to purely object-oriented code: (1) annotations can have positive effects on program comprehension; (2) decomposition can have



### KEYWORDS

Program comprehension, Feature traceability, Software maintenance, Separation of concerns

**ACM Reference Format:**
Jacob Krüger, Gül Çalıklı, Thorsten Berger, Thomas Leich, and Gunter Saake. 2019. Effects of Explicit Feature Traceability on Program Comprehension. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3338906.3338968


### 1 INTRODUCTION

Developers often need to understand the purpose and the details of specific parts of a codebase, which is a time-consuming and cognitively demanding activity during software engineering [32, 59, 60]. A developer performs this activity, known as *program comprehension*, when they are new to a program or forgot details that are required for their task [8, 30]. Consequently, to gain implicit knowledge about a program, developers need to read and comprehend the

# Research Goal

- Numerous techniques to improve program comprehension and trace features.
- Often heavyweight or separated from actual code.
- Can explicit feature traceability on code level support program comprehension?
  - Annotations
  - Components

# Research Questions

**RQ1:** What is the impact of feature traces on effectively solving tasks?

**RQ2:** What is the impact of feature traces on efficiently solving tasks?

**RQ3:** What is developers' perception of feature traces?

# Methodology: Online Experiment

- 49 participants
- Three tasks on feature comprehension, three on bug localization
- Measured time and correctness
- Questions on participants' perception

**Treatment Group 1**

Feature traceability with **annotations**

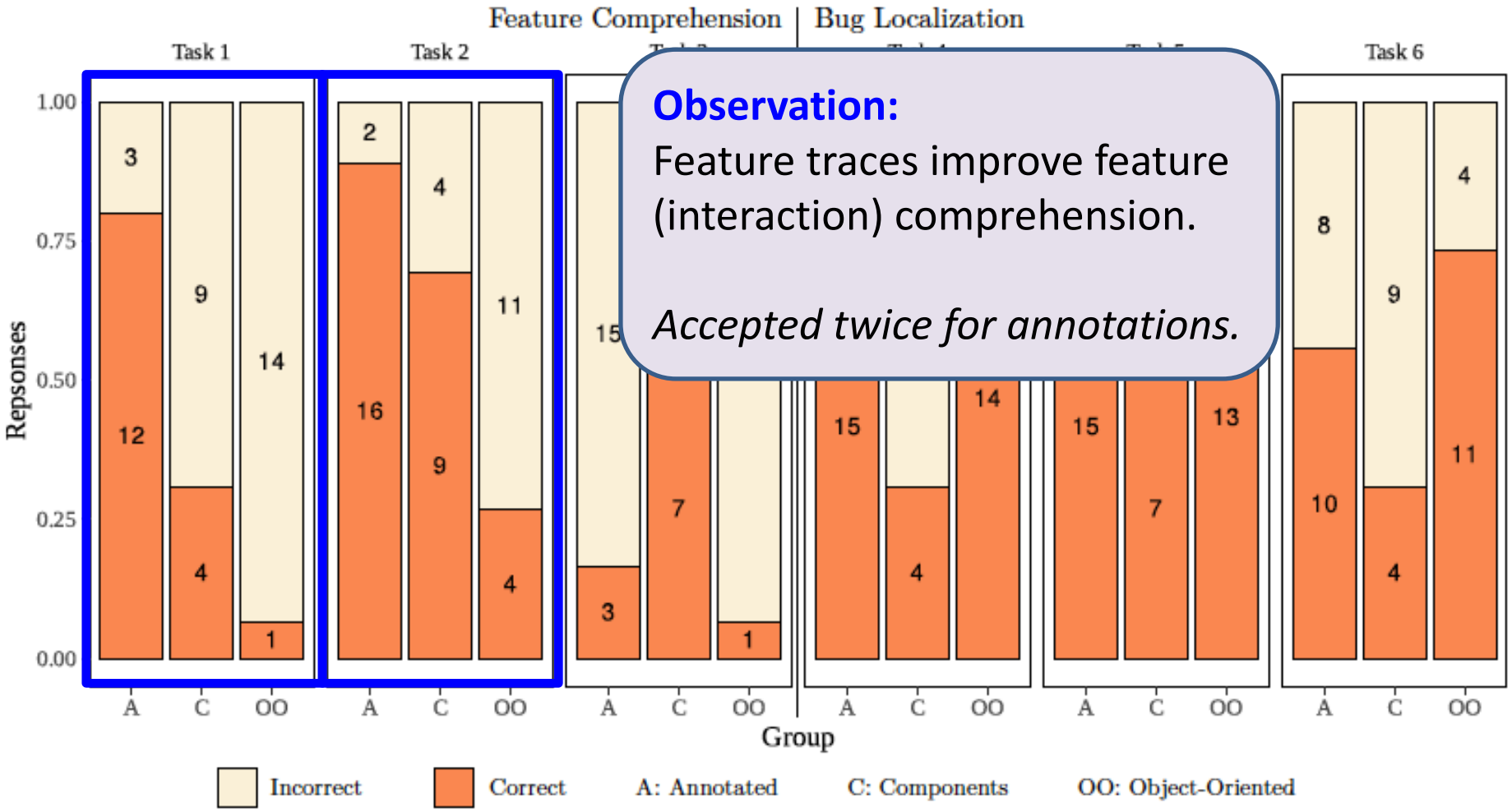
**Treatment Group 2**

Feature traceability with **components**

**Control Group**

NO feature traceability **(object-oriented)**

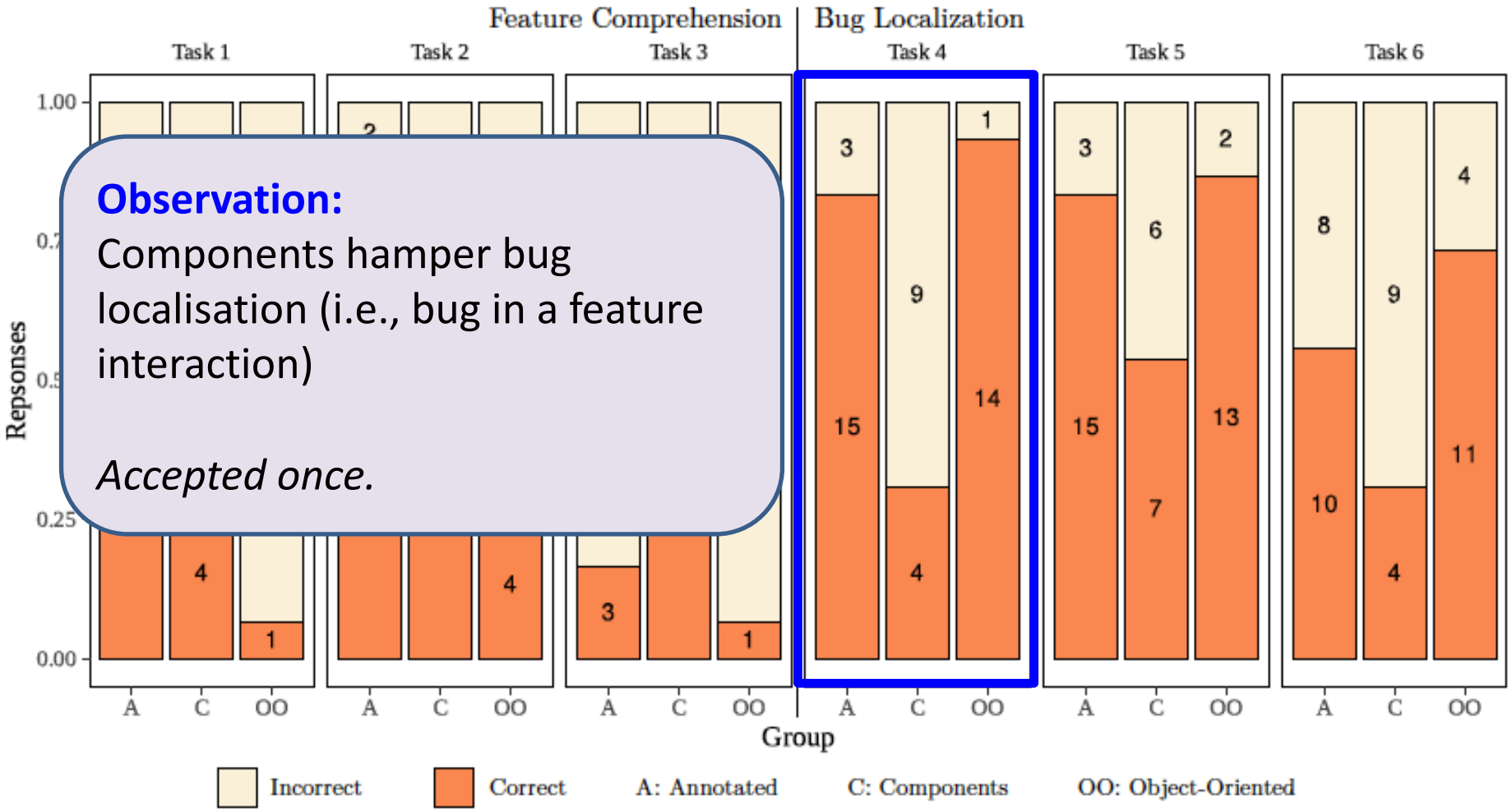
# Results: Effectiveness (RQ1)




**Observation:**
Feature traces improve feature (interaction) comprehension.

*Accepted twice for annotations.*

# Results: Effectiveness (RQ1)




**Observation:**
Components hamper bug localisation (i.e., bug in a feature interaction)

*Accepted once.*

# Results: Efficiency (RQ2)

**Observation:**
Explicit feature traces do not impact the analysis time.

*Not rejected.*

| | Task 1 | | | Task 2 | | | Task 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | C | OO | A | C | OO | A | C | OO |
| Und. Part. | 10 | 10 | 9 | 13 | 12 | 15 | 16 | 14 | 15 |
| Incl. Part. | 10 | 8 | 9 | 12 | 11 | 13 | 14 | 14 | 13 |
| Times (mins) | | | | | | | | | |
| Min | 2.91 | 2.23 | 2.72 | 0.44 | 1.14 | 0.91 | 0.70 | 0.67 | 0.52 |
| Mean | 13.07 | 5.51 | 12.27 | 1.72 | 3.26 | 3.30 | 2.73 | 2.26 | 1.84 |
| Median | 11.23 | 4.03 | 9.75 | 1.06 | 2.63 | 2.09 | 2.04 | 2.11 | 1.68 |
| Max | 25.02 | 12.73 | 22.92 | 4.90 | 8.48 | 11.96 | 7.29 | 4.70 | 3.90 |
| SD | 8.34 | 3.59 | 7.54 | 1.43 | 2.34 | 3.14 | 1.78 | 1.30 | 0.89 |

Part.: Participants; Und.: Undisturbed; Incl.: Included; SD: Standard Deviation

# Results: Perception (RQ3)

*"Yes, they did. In fact, without the annotations (provided that they are correct), it would have been significantly more difficult to understand which part of the code does what."*

| | | # Mentioned | |
|---|---|---|---|
| Get picture of code | 7 | 6 | 12 |

*"[N]o, adding comments is a bad sign, it screams that code is not self explanatory enough."*

| | | | |
|---|---|---|---|
| Follow class names | – | 7 | – |

*"On the one hand, it made the classes small and locating possibly relevant code easy. On the other hand, interactions were more difficult to spot, because I had to switch between different classes."*

| | | | |
|---|---|---|---|
| Comments | – | 0 | 4 |
| Explicit locations | – | – | 3 |

# Results: Perception (RQ3)

| Response | # Mentioned | | |
|---|---|---|---|
| | Annotations | Components | Object-Oriented |
| | | is strategy | |
| | | | 12 |
| | | | 8 |
| | | | 3 |
| | | | – |
| | | | – |
| | | Code design | |
| Positive | 14 | 9 | – |
| Unsure | 2 | 2 | – |
| Negative | 2 | 3 | – |
| Components | 1 | – | 5 |
| Comments | – | 0 | 4 |
| Explicit locations | – | – | 3 |

**Observations:**

Explicit feature traces:

- extend analysis strategies,
- are unproblematic to use, and
- are positively perceived.

# Conclusions

- Annotations have positive impact on program comprehension.
- Components can negatively impact bug localization:
  - Depends on the decomposition strategy
  - Requires analysis at what point a component is useful
- Feature traces do not impact analysis efficiency.
- Feature traces are understandable and positively perceived.

→ Annotations seem proper to introduce feature traceability in practice

# To Conclude …

# Summary of the Talk

## Cognitive Biases in SE: Research Gap

**Research Gap #1:**

Is the observed phenomenon manifestation of the claimed cognitive bias?

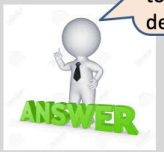
CHALMERS | UNIVERSITY OF GOTHENBURG 11

## Cognitive Biases in SE: Research Gap

**Research Gap #2:**

Mediating processes that manifest cognitive biases (e.g., What happens in memory, working memory, etc.?)

Why is understanding mediating processes important?




It can help development of tools/techniques for de-biasing.

CHALMERS | UNIVERSITY OF GOTHENBURG 38

## Application of Proposed Solution

accepted at ICSE'20

**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

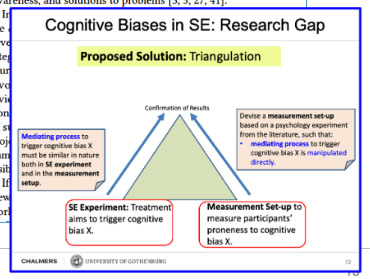
Gül Çalikli
gul.calikli@gu.se
Chalmers & University of Gothenburg
Gothenburg, Sweden

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

development teams by means of improved knowledge transfer, awareness, and solutions to problems [3, 5, 27, 41].




CHALMERS | UNIVERSITY OF GOTHENBURG

## Towards Understanding Working Memory…

ESEC/FSE 2019

**Effects of Explicit Feature Traceability on Program Comprehension**

Jacob Krüger
Otto-von-Guericke University
Magdeburg, Germany
jkrueger@ovgu.de

Gül Çalıklı
Chalmers | University of Gothenburg
Gothenburg, Sweden
calikli@chalmers.se

Thorsten Berger
Chalmers | University of Gothenburg
Gothenburg, Sweden
bergert@chalmers.se

Thomas Leich
Harz University & METOP GmbH
Wernigerode & Magdeburg, Germany
tleich@hs-harz.de

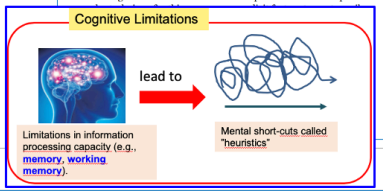
Gunter Saake
Otto-von-Guericke University
Magdeburg, Germany
saake@ovgu.de

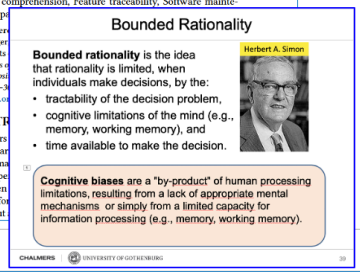
**ABSTRACT**

Developers spend a substantial amount of their time with program comprehension. To improve their comprehension and refresh their memory, developers need to communicate with other developers, read the documentation, and analyze the source code. Many studies show that developers focus primarily on the source code and that small improvements can have a strong impact. As such, it is crucial to bring the code itself into a more comprehensible form. A partic-

**KEYWORDS**

Program comprehension, Feature traceability, Software maintenance, Sep



# References


**[3]** A. Tang, ”Software designers, are you biased?" in Proceedings of the 6th International Workshop on Sharing and Reusing Architectural Knowledge. ACM, 2011, pp. 1-8.

**[4]** K. Mohan and R. Jain, ”Using traceability to mitigate cognitive biases in software development," Communications of the ACM, vol. 51, no. 9, pp. 110-114, 2008.

**[5]** K. Mohan, N. Kumar, and R. Benbunan-Fich, ”Examining communication media selection and information processing in software development traceability: An empirical investigation," IEEE Transactions on Professional Communication, vol. 52, no. 1, pp. 17-39, 2009.

**[6]** T. K. Abdel-Hamid, K. Sengupta, and D. Ronan, ”Software project control: An experimental investigation of judgment with fallible information," IEEE Transactions on Software Engineering, vol. 19, no. 6, pp. 603-612, 1993.

**[7]** W. Stacy and ]. MacMillan, ”Cognitive bias in software engineering," Communications of the ACM, vol. 38, no. 6, pp. 57—63, 1995.

**[8]** G. Browne and V. Ramesh, ”Improving information requirements determination: a cognitive perspective," Information 8 Management, vol. 39, no. 8, pp. 625-645, 2002.

**[9]** J. A. O. da Cunha, F. Q. da Silva, H. P. de Moura, and F. J. Vasconcellos, ”Decision-making in software project management: A qualitative case study of a private organization,” in Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering. ACM, 2016, pp. 26-32.

**[10]** J. Parsons and C. Saunders, ”Cognitive heuristics in software engineering applying and extending anchoring and adjustment to artifact reuse," IEEE Transactions on Software Engineering, vol. 30, no. 12, pp. 873-888, 2004.

**[11]** C. Mair and M. Shepperd, ”Human judgement and software metrics: Vision for the future," in Proceedings of the 2nd international workshop on emerging trends in software metrics. ACM, 2011, pp. 81-84.

**[12]** K. A. De Graaf, P. Liang, A. Tang, and H. Van Vliet, ”The impact of prior knowledge on searching in software documentation,” in Proceedings of the 2014 ACM symposium on Document engineering. ACM, 2014, pp. 189-198.

**[13]** R. Jain, ]. Muro, and K. Mohan, ”A cognitive perspective on pair programming,” AMCIS 2006 Proceedings, p. 444, 2006.

**[14]** ]. A. O. G. da Cunha and H. P. de Moura, ”Towards a substantive theory of project decisions in software development projectbased organizations: A cross-case analysis of it organizations from brazil and portugal," in Information Systems and Technologies (CISTI), 2015 10th Iberian Conference on. IEEE, 2015, pp. 1—6.

**[15]** G. Calikli, A. Bener, B. Caglayan, and A. T. Misirli, ”Modeling human aspects to enhance software quality management,” in Thirty Third International Conference on Information Systems, 2012.

# References

**[16]** M. Nurminen, P. Suominen, S. Ayramo, and T. Karkkainen, ”Applying semiautomatic generation of conceptual models to decision support systems domain," in IASTED International Con— ference on Software Engineering (SE 2009). ACTA Press, 2009.

**[17]** G. Calikli and A. Bener, ”Empirical analyses of the factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering. ACM, 2010, p. 10. G. Calikli, A. Bener, and B. Arslan

**[18]** L. M. Leventhal, B. M. Teasley, D. S. Rohlman, and K. Instone, ”Positive test bias in software testing among professionals: A review," in International Conference on Human—Computer Interaction. Springer, 1993, pp. 210—218.

**[19]** E. D. Smith, Y. ]. Son, M. Piattelli-Palmarini, and A. Terry Bahill, ”Ameliorating mental mistakes in tradeoff studies," Systems Eh— giheerihg, vol. 10, no. 3, pp. 222—240, 2007.

**[20]** G. Calikli, A. Bener, T. Aytac, and O. Bozcan, ”Towards a metric suite proposal to quantify confirmation biases of developers," in Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on. IEEE, 2013, pp. 363—372.

**[21]** F. Shull, ”Engineering values: From architecture games to agile requirements," IEEE Software, vol. 30, no. 2, pp. 2—6, 2013.