DOI:10.1145/3517217

To view the accompanying Technical Perspective, visit doi.acm.org/10.1145/3517215

Cognitive Biases in Software Development

By Souti Chattopadhyay, Nicholas Nelson, Audrey Au, Natalia Morales, Christopher Sanchez, Rahul Pandita, and Anita Sarma

Abstract

Cognitive biases are hardwired behaviors that influence developer actions and can set them on an incorrect course of action, necessitating backtracking. Although researchers have found that cognitive biases occur in development tasks in controlled lab studies, we still do not know how these biases affect developers' everyday behavior. Without such an understanding, development tools and practices remain inadequate. To close this gap, we conducted a two-part field study to examine the extent to which cognitive biases occur, the consequences of these biases on developer behavior, and the practices and tools that developers use to deal with these biases. We found about 70% of observed actions were associated with at least one cognitive bias. Even though developers recognized that biases frequently occur, they are forced to deal with such issues with ad hoc processes and suboptimal tool support. As one participant (IP12) lamented: *There is no salvation!*

1. INTRODUCTION

Cognitive biases are systematic deviations from optimal reasoning[17] that influence how we find, evaluate, and remember information. These "shortcuts" to potential solutions can take several forms, and regularly occur in our everyday behavior. For example, confirmation bias (*tendency to pay more attention to information that agrees with our preconceptions*) is demonstrated in some individuals' characterization of the COVID-19 virus as "just another u," prompting them to engage in social behavior contrary to experts' and health organizations' advice. As with this example, the occurrence of cognitive biases can cause significant impacts on society.

Software developers are not immune from such behavior, and may exhibit biases for several reasons. For example, some biases are a result of attempts to bypass our limited cognitive capacity (for example, availability bias may prompt developers to choose solutions based on examples they readily remember), whereas others are a result of prior experience with a solution (for example, belief perseverance bias may force developers to focus more on the code *they believe* has the bug), or individual problem-solving styles (for example, hyperbolic discounting bias may encourage developers to choose a solution with smaller and quicker rewards).

Although these cognitive biases often result in desired solutions, they may also cause significant negative consequences. Controlled lab studies have identified the harmful effects of specific cognitive biases on several aspects of software development such as defect density,[1] requirements specification,[6] originality of design,[18] and feature design.[4] Mohanani et al.[9] conducted a survey of 65 such studies. However, despite these efforts, we still do not understand how cognitive biases manifest in the real world, and how they influence developer actions, behavior, and decision-making *in situ*. Only by understanding real-world behavior, we can begin to understand how to curtail such nonoptimal behavior.

Here we present results from a two-part field study examining to what extent cognitive biases occur in a developer's daily work activities. We consider how frequently, and when, certain biases occur. We also list some current practices and tools that developers currently use to mitigate biases.

2. BACKGROUND

Cognitive biases were first introduced in 1974 by researchers Tversky and Kahneman.[17] Researchers in software engineering have been studying these biases in this domain since 1990.[16] Mohanani et al. summarizes 65 articles that characterize the current state of studying biases in software engineering.[9] These articles investigate 37 distinct cognitive biases (out of more than 200 previously identified in psychology, sociology, and management research). For example, studies have found that confirmation bias leads to higher defect rates and more postrelease defects when testing; availability and representativeness biases lead to developers misrepresenting code features; and overconfidence caused insufficient efforts when performing requirements analysis.

Of the 65 papers examined, none describe the use of *in situ* field studies as part of their research methodology. For example, Calikli et al.[2] evaluate the effects of company culture, education, and experience on confirmation bias among software developers and testers through user studies involving interactive and written tests. Although lab studies and controlled experiments allow for control of confounding factors, they do sacrifice the richness and spontaneity of naturalistic observations.[5] Our study attempts to extend these lab-based (or studies in nonnatural environment) findings to actual development practice via the use of observational studies in a real-world setting.

3. METHODOLOGY

We observed 10 developers *in situ* for our field study. Our participants were recruited from a U.S.-based software

The original version of this paper is entitled "A Tale from the Trenches: Cognitive Biases and Software Development" and was published in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (pp. 654–665), June 2020.

startup (company A) that specializes in the areas of distributed developer tools and services, such as program analysis, user interface (UI) design, infrastructure support, and software R&D. Due to the diversity of focus areas in the startup, participants used a wide variety of programming languages, tools, and working styles.

Table 1 presents demographic information about study participants, such as development experience, code editor usage, and preferred programming languages (the median software development experience was two years, and the mean was 5 years 9 months).[a]

We observed participants performing their routine development tasks on a typical workday. During the observation session, we asked participants to think aloud and verbalize their thoughts and interactions.[12] We recorded their screen, audio, and physical workspace.

The total observation time per session was limited to 60 min to prevent participant burnout and respect time restrictions at the startup.

During each session, one researcher was positioned next to the participant, taking notes. In a separate room, not visible to the participant, an additional researcher served as a secondary field-note taker, and monitored the recordings of the participant to ensure consistency. Members of the research team alternated between serving as the primary and secondary note takers.

At the end of each session, participants were asked to complete a brief demographic survey (see results in Table 1) although the two researchers compiled and prepared follow-up questions to clarify in a 15-min retrospective interview.

To understand how cognitive biases affect software development, we first identified developer tasks (goals) and *actions*—discrete steps performed by the developer to reach the goal.[3] We identified and classified each individual action taken by participants using qualitative coding methods.

a We could not study whether gender had any association with biases as very few participants in our study were women.

Table 1. Study participant demographics.

Ptc.[i]	Gnd.[ii]	Exp.[iii]	Language(s)[iv]	Editor[v]
P1	M	21 years 0 months	Java	Eclipse
P2	M	1 year 11 months	Clojure	Eclipse
P3	M	1 year 10 months	Clojure, Java	Emacs
P4	M	7 years 3 months	Clojure, Python	Emacs
P5	M	2 years 0 months	Clojure, Java, Haskell	Emacs
P6	M	2 years 0 months	TypeScript, Clojure, Java	VS Code
P7	M	5 years 0 months	C/C++	Emacs
P8	F	15 years 0 months	JavaScript, CSS	VS Code
P9	M	0 years 9 months	C, Prolog	Sublime
P10	F	1 year 0 months	Python	PyCharm

[i] Ptc. = Participant.
[ii] Gnd. = Gender.
[iii] Exp. = Software development experience.
[iv] Preferred language(s).
[v] Editor used in session.

To code the raw data, we first transcribed all data and partitioned each unit to contain a quote and description of the participant's actions aligned with the timestamp.

To code the actions in our transcript data, we created five sets of 94 (4.5%, total 22.5% over five sets) random instances from the observations across all participants. Three authors individually coded each action with the codes described in Table 2. We achieved inter-rater reliability (IRR) measures for three coders. We then examined each action to identify any associated biases as described next.

3.1. Complementary interviews

We next conducted semistructured interviews with 16 developers to triangulate our findings from the initial field study. We used semistructured interviews instead of surveys to: (1) confirm that participants correctly understood the biases, although allowing them to ask clarifying questions, and (2) follow up on advice/practices that participants suggested to address bias.

We recruited interview participants from three companies to examine debiasing practices across a variety of organization sizes and cultures. First, we interviewed 11 developers from the original company in our field study (Company A); from our observed 10 participants (see Table1), two employees had left and three others had joined since our field study. Next, we interviewed one developer from another start-up of similar stature (Company B); team sizes were similar with those at Company A. Finally, we interviewed four developers from a Fortune 500 company (Company C); a multinational company with large team size. These interviews helped to confirm that the observed biases were not limited to a single company. Table 3 provides demographic information for all of the interview participants.

In the interviews, we defined 10 bias categories and provided examples based on instances observed in our field study (see Section 4 for definitions). Using these generalized examples, we asked two questions for each specific bias: (1) "On a scale from 1 (low) to 5 (high), how often do you think developers act under this bias?" and (2) "What standard practices, guidelines, or tools would help to avoid this bias?"

Interview responses were categorized by two authors using Pattern Coding[8]—the process of grouping categories into smaller thematic sets. We identified 29 development practices (for example, brainstorming, referencing). These practices were abstracted into five categories that link specific biases with practices that directly address them (see Table 6 for details).

Table 2. Action codes.

Action	Definition
Read	Examining information from artifacts (for example, code, documentation, terminal output)
Edit	Any change made directly to code or artifacts
Navigate	Moving within or among artifacts (for example, pulling files from Git, opening files, scrolling through a file)
Execute	Compiling and/or running code
Ideate	Constructing mental model of future changes

Table 3. Interview participant demographics.

Pt[i]	Gn[ii]	C[iii]	Exp[iv]	Role[v]	Pt	Gn	C	Exp	Role
IP1	M	A	23 years 0 months	Dev	IP9	M	A	8y 0 months	Dev
IP2	M	A	2 years 11 months	Dev	IP10	M	A	2 years 0 months	Dev
IP3	M	A	2 years 10 months	Dev	IP11	M	A	1 year 9 months	Dev
IP4	M	A	8 years 3 months	Dev	IP12	M	B	1 year 0 months	Dev
IP5	M	A	3 years 0 months	Dev	IP13	M	C	5 years 0 months	Dev
IP6	F	A	19 years 8 months	QA	IP14	F	C	2 years 0 months	Dev
IP7	M	A	6 years 0 months	Dev	IP15	M	C	2 years 0 months	Dev
IP8	M	A	19 years 0 months	Dev	IP16	M	C	5 years 0 months	Dev

[i] Pt. = Participant.
[ii] Gn. = Gender.
[iii] C. = Company.
[iv] Exp. = Years/months of software development experience.
[v] Job position in the company.

Table 4. Cognitive bias categories.

	Bias category	Bias(es)	Example
CB1	Preconceptions	Confirmation, selective perception	P1 continually added hashmaps when other data structures were more suited for data query APIs.
CB2	Ownership	IKEA effect, endowment effect	P8 decided to reuse her old CSS file instead of the premade CSS files from the Bootstrap project.
CB3	Fixation	Anchoring and adjustment, belief preservation, Semmelweis reflex, fixation	P9 fixated on changing the function definitions when the environment just needed to be reloaded.
CB4	Resort to Default	Default, status-quo, sunk cost	P2 opened a new code file and kept unused template code at the top of the file.
CB5	Optimism	Valence effect, invincibility, wishful thinking, overoptimism, overconfidence	P4 was proud of his new aggregating map code, but it got an error after it was printed.
CB6	Convenience	Hyperbolic discounting, time-based bias, miserly information processing, representativeness	P2 created simple overly-verbose code that addressed his current needs, but became spaghetti code that slowed future progress.
CB7	Subconscious action	Misleading information, validity effect	P6 focused on fixing the files listed in error messages instead of the core dependency file causing errors throughout the system.
CB8	Blissful ignorance	Normalcy effect	P10 disregarded all compiler warnings out of habit and failed to notice a new exception detailing the cause of his build failure.
CB9	Superficial selection	Contrast effect, framing effect, halo effect	P4 copied and pasted a function from his documentation directly into his syntax without examining it first.
CB10	Memory bias	Primacy and recency, availability	P1 reused a design pattern that worked well on recent tasks, because he could easily recall the structure of the code.

4. BIAS CATEGORIZATIONS

A description of individual biases can be found in our supplemental site,[b] which provides anonymized supplemental artifacts used for data analysis. We cannot release raw data due to participant privacy concerns.

4.1. Bias categories

We grouped the *28 observed biases into 10 categories* based on their effect on developer behavior (see Table 4); we did not observe the remaining nine biases reported by Mohanani et al., likely because of our study design—an hour-long observational study with think aloud protocol.

The bias categories were created through the process of negotiated agreement to ensure the validity of the bias coding. Two authors individually categorized the 28 biases using the following: (1) the definitions of cognitive biases in the context of software engineering (per Mohanani et al.), (2) the definitions of biases in cognitive science literature, and (3) the observed effects of biases on participants' development behavior (through direct observation and participants' verbalization). In the first round, the authors agreed on the categorization of 24 out of 28 biases (85.7% agreement), into a set of 11 categories. In the second round, the authors disagreed on one bias categorization (96.4% agreement) and decided to merge the 1st and 11th categories. Table 4 shows the final list of 10 categories (CB1–CB10), and their mapping to individual cognitive biases.

The *Preconceptions* (CB1) category refers to the tendency to select actions based on preconceived mental models for the task at hand. Biases within this category cause developers to discount the degree of solution space exploration required to take action.

Ownership (CB2) occurs when developers give undue weight to artifacts that they themselves create or already possess, thereby reducing the potential for other options to be objectively evaluated. Preference for one's own artifacts

b https://epiclab.github.io/ICSE20-CogBias/

prevents developers from exploring the solution space completely.

Fixation (CB3) refers to anchoring problem-solving efforts on initial assumptions, and not modifying said anchor sufficiently in light of added information or contradictory evidence. This leads to reduced awareness of task context.

Resort to default (CB4) occurs when developers choose readily available options based solely on their status as the default, or the tendency to prefer current conditions without regard to applicability or fitness. This causes lost context of the overall task.

Optimism (CB5) reflects the set of biases that lead to false assumptions and premature conclusions regarding efficiency or correctness of a chosen solution. This occurs when people over-trust their abilities, or when the likelihood of a favorable outcome is overestimated.

Convenience (CB6) encompasses the assumption that simple causes exist for every problem, and the predisposition to take the seemingly quicker or more simplistic routes to solution. This reduces the effort developers invest in reasoning and making sense of information.

Subconscious action (CB7) refers to the offloading of evaluation and sense-making to external resources (such as IDEs or online resources) without regard to the actual merits of such information.

Blissful ignorance (CB8) refers to the assumption that everything is nominal and working, even in the face of information indicating otherwise. Because of this, developers do not pay attention to their surroundings.

Superficial selection (CB9) represents a range of actions and information being unduly valued based on superficial criteria. As a result, developers decide on a solution without thoroughly reasoning through it.

Memory bias (CB10) affects how developers remember information from a series of alternates, prefer to use the primary or most recent information encountered, or react as a result of information most readily available in the memory.

5. RESULTS

5.1. Presence of biases in developer actions

The field study included 2084 distinct developer actions; of these we classified 953 actions that contained at least one bias category. Thus, approximately half of developer actions (45.72%) were associated with some form of bias. Note, the large number of biased actions (953 out of 2084) in our observation may likely be due to cognitive biases being inherent in decision-making actions, which are a key part of software development.

However, not all cognitive biases necessarily result in a negative outcome. Biases can lead to positive effects—participants taking fewer actions than anticipated. However, in a noncontrolled environment, we cannot differentiate between the "baseline" (no-bias) or "optimized" (positive outcomes of bias) number of actions. Thus, here we focused only on reversed actions (negative bias).

To identify biases that resulted in a negative outcome, we use the notion of **`Reversal Actions`**. We define **`Reversal Actions`** as the actions that developers need to undo, redo, or discard at a later time. Reversal actions are thus indicative of nonoptimal solution paths.

Figure 1a shows the distribution of developer actions (biased or nonbiased), and whether it led to a negative outcome. There were 953 actions with biases, and 1131 without. Similarly, there were 1104 reversal actions and 980 nonreversal actions. Reversal actions were more likely to occur with a bias—68.75% (759/1104 cases), and biased actions were more likely to be reversed—79.64% (759/953 cases).

To verify this association, we conducted a chi-square test of independence with a Bonferroni correction (to account for multiple comparisons[13]). The chi-square test is significant ($\chi^2[4, N = 2084] = 499.35$, p-value < $2.2e-16$, with Bonferroni correction) showing *biased actions were highly associated with reversals*.

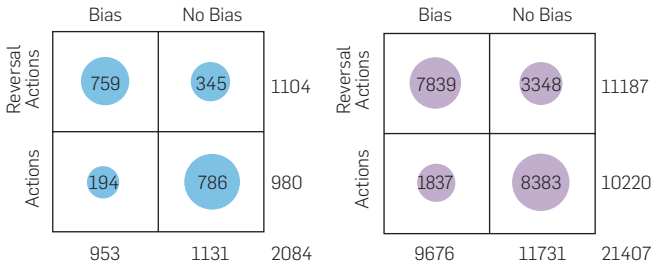
To evaluate the strength of this association, we estimated the Cramer's V measure that signifies a strong association between the presence of bias and actions that need to be reversed (V= 0.5, large when min. number of variables is 2[14]).

However, if the time spent on reversing actions is not substantial, the number of reversal actions alone does not provide an accurate estimate of the negative outcomes of biases. We analyzed the time spent during each action, captured in Figure 1b. Each cell presents the time (in second) spent in each type of action.

In total, developers spent 34.51% (7839/21407) of their time reversing–biased actions. When focusing only on the time spent in reversal actions, 70.07% (7839/11187) of these involved biases. Therefore, biased actions lead to significant negative outcomes as participants lost approximately 25% of their entire working time. A chi-square test of independence supports this hypothesis, $\chi^2[4, N = 21407] = 5850.2$, p-value < $2.2e-16$ showing time spent reversing actions is not independent of biased actions (with Bonferroni correction and large effect size with Cramer's V).

Not only are biases frequent in software development (45.72%), but also biased actions are significantly more likely to be reversed. Also, developers spend a significant amount of time reversing these biased actions.

Figure 1. Distribution of presence of bias and reversal actions. Size of circles represent (a) the number of actions or (b) time (in seconds). Totals are shown along the bottom and right edges, with overall totals shown in the lower right-hand corners.



(a) Distribution of Actions (b) Distribution of Time (sec)

5.2. Consequences of biases

To identify the consequences of these biases on development, we investigated the effects of bias categories on participants' decision-making and problem-solving.

Two authors categorized the effects of the bias categories into four consequence groups by analyzing the biases and using negotiated agreement. Table 5 shows the consequences of biases on development and the associated categories.

We identified the effect of each bias category on four orthogonal problem-solving activities in programming: (1) gathering information,[15] (2) making sense of the information,[11] (3) maintaining information (context) that is relevant to tasks and goals,[3] and (4) maintaining and focusing attention in the necessary places.[10]

Inadequate exploration. Exploring or foraging different pieces of information[10] and evaluating alternate solutions[7] form a key part of development. Cognitive biases sometimes inhibited participants from investing in proper exploration.

Reduced explorations often led to participants creating suboptimal solutions. For example, P4 needed a subset of data from a hashmap which required him to query the hashmap. As he was not familiar with the query interface and did not know how to construct the query, he decided that an easy-fix (CB6) was to instead manually collect the data.

[14:26]*"Easiest thing to do would be to collect all input statements and instead of using the query, do this myself."*

P4 then began implementing this functionality under the preconception (CB1) than manual data collection is easy. However, after trying this for the next 18 min, he realized that the implementation was far more difficult than what he had expected, at which point he decided to learn how to query a hashmap.

Reduced sense-making. Sense-making is the process of cognitively engaging with information to construct a relevant mental model, which can then be used to understand a given situation.[11] We identify reduced sense-making through participants' verbalization indicating that previous actions (and assumptions) were incorrect.

For example, P10 was testing modifications to data pipelines (used to aggregate and monitor data) and found that her tests failed after she added new input data files to the pipeline. She subconsciously (CB7) followed the error location suggested in the message without reasoning about the error. Eventually, she found that she was using older input files which caused the error; her tests worked after she updated these files.

Table 5. Consequences of biases.

Consequence	Bias categories (CB)
Inadequate exploration	1, 2, 4, 5, 6, 10
Reduced sense-making	5, 6, 7, 8, 9
Preserving context	3, 5, 8
Misplaced attention	3, 4, 8, 9

[13:25]*"... So that one [file] wasn't there to begin with ..."*

Context loss. When navigating and making sense of different sets of information, developers must retain a mental model of the problem space and relevant information to complete a task.[3] A reduction in context can be seen when participants repeatedly backtrack or verbalize confusion regarding the current task or goal (for example, losing track of their current actions).

This is shown when P3 fixates (CB3) on trying to solve an error and gets sidetracked, thus losing the larger context of their task. We also observed that when participants were optimistic (CB8) about an implementation, they would suspend the related context and move on with their task. These participants struggled to recall the context at a later time when their implementation failed.

Misplaced attention. Attention is a critical element of our cognitive system, and affects what information developers perceive as relevant, how developers interpret error messages, and so forth. Biases can cause developers to misplace their attention causing them to spend time working on issues that are irrelevant to the current task.

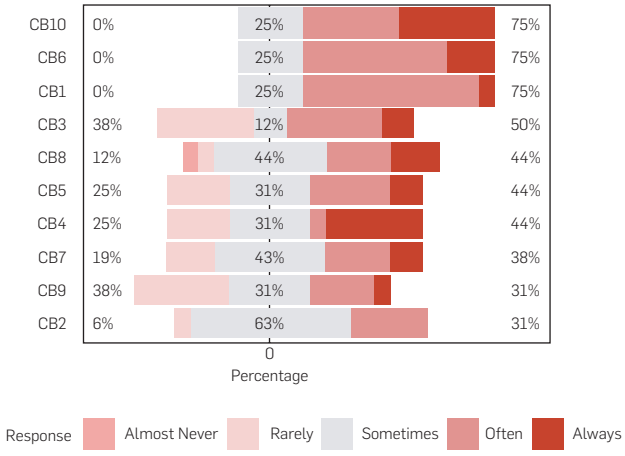
When P4 tried to debug his query function which was returning `nil`, he thought the problem could be an incorrect query syntax.[26:28]*"This is the API for Clojure and I'm looking for something that tells me how to check if a list contains a* `vect`*."* He became so focused on changing his syntax (CB3) that he did not notice the syntax highlighting was no longer working and his environment had failed (CB8).

Biases affect multiple aspects of problem-solving during development. Specifically, biases affect how adequately developers explore the solution space, how thoroughly they engage in sense-making, how effectively they retain context, and how efficiently they invest their attention.

5.3. Dealing with biases

To add further evidence that biases occur frequently in practice, we interviewed 16 developers asking them: "how often

Figure 2. Perceived frequency of biases from interviews ranging from "Almost Never" to "Always." Bias categories are ordered in descending order from most to least frequent (shown in %).



do you think developers act under [bias category]?"

Figure 2 shows the perceived frequency of occurrences for each bias; ranging from "Almost never" to "Always." The frequencies are indicated by hues of red. Dark red bars denote high frequencies (Often, Always), with their percentages reported far right (for example, 81% of interviewees considered CB10 to be frequently occurring). Light red bars denote low frequencies (Almost Never, Rarely), with percentages reported far left (for example, 6% found CB10 to be infrequent). Grey bars in the center reflect the frequency of "sometimes" responses, which is considered neutral within subsequent analysis.

Overall, interviewees perceived biases to occur frequently in software development (Figure 2), matching our observations. For example, when talking about Convenience bias (CB6), IP12 said:[32:12] *"It happens all the time! … It's the story behind why technical debt happens! Three months and then you go and ask why on earth is this failing? And when you look back and somebody overwrote something because it was easier. And it screwed up everything!"*

Memory (CB10), Convenience (CB6), and Preconception (CB1) bias were ranked highest in perceived frequency. These ratings are, at least, partial confirmation of our empirical findings, as Convenience (CB6), Fixation (CB3), and Preconception (CB1) were likewise in the top five most observed biases. However, for Memory bias (CB10) and Subconscious Action (CB7), the actual frequencies do not match with developers' perceived frequency.

Practices that help. Although current development practices and tools are not designed to avoid cognitive biases, developers might still be using them to do so. Therefore, during our interviews, we asked participants to identify practices and tools that could help them/coworkers avoid or recover from biases. There were 246 unique suggestions in the interviews. Two authors categorized these suggestions using Pattern Coding[8]—the process of grouping categories into smaller sets of themes. Three themes emerged—Development Practices, Who performs these practices, and When.

Table 6 displays these categories and themes, along with the biases that these practices can help with. The last column lists the tools that participants found useful to help with these practices. The "Categories" column indicates the category of the practice, and the "Subcategories" column describes helpful practices within the category. The "Who" column indicates whether the team (T) needs engage in the practice or the individual (I) can do it themselves. The "When" column specifies whether the practice needs to be done Before (B), During (D), or After (A) a task. The "Biases" column indicates the biases each practice can help with.

The five categories of suggestions are as follows.

Stepping back: Taking a break from one's own development pattern can help developers become aware of beneficial practices (such as clean code), which can avoid biases such as Preconception (CB1) and Memory (CB10). For example, IP13 described *Documentation days* and *Test Fests*, as: [20:01] *"… documentation days are where we say, 'Today, we're not going to be writing code. We're going to focus on checking the documentation, updating documentation …' You might run into some of the new methods … and then you are more bound to use them next time."*

Similarly, learning through focused and incentivized training can help ingrain "good" practices that will help developers avoid biases such as Convenience (CB6). For example, IP14 mentioned how "clean code" workshops were: [20:06] *"really instilled in all of us—oh, it really matters to build the highest quality code!"*

Table 6. Helpful practices.

Categories	Subcategories	Time	Who	Biases	Tools
Stepping Back	**Incentivized training:** discussion of clean code benefits, long-term goals	B/D	T/I	2,6,7	NA
	Noncode days: documentation days, test fest, familiarize with concepts	D	T	1,6,10	
	Meaningful configurations, updating configurations, meaningful defaults	B	T	4,10	
Different Perspectives	**Confer with developers:** pair programming, collaborative brainstorming (code/design/tool), verify global changes, designated tool guy	B/D	T	1,2,3,6,9,10	Slack, Hipchat
	Open communication: encourage communication, communicate early with teams such as QA. Promote focus on functionality and need	D	T	2,5,6,7	
Systematic Approach	**Systematic exploration:** prior research on tools, compare and contrast solutions, problem decomposition	B	I	1,2,3,4,5,6,7,8,9	Dev Tools, Sonarlint
	Big picture in mind: reusability of code, backward compatibility	D	I	1,2,6,10	
	Consistent early feedback: reviews (design, expert, peer), sprint meetings	D/A	T	1,2,3,4,5,6	
RTFM	**Reference doc.:** req/API/design doc, code comments, online sources	B	T	1,4,5	IDE Suggestion
	Journal options/alternatives: playbook, team diary	B	T/I	1,2,10	
	Meaningful and relevant specifications: standard specs, severity/ relevant levels for warnings, protocol for resolution of warnings, descriptive errors	B/D	T	2,4,7,8	
Processes	**SE concepts:** agile, code review, shared artifacts, reduced ownership, design first, UML diagrams, user story, TDD, BDD, constant debugging, data flows	B/D/A	T/I	1,3,4,5,6,7,8,9,10	ZenHub, Gerrit, Debugger, IDE,
	Standardization: corporate/coding/package standards, right arch. and microservices, clean code, performance test, impact analysis	A	T	1,2,5,6,7,8,9,10	JIRA, JaCoCo
	Problem-solving strategies: divergence and convergence thinking, defensive programming, negative hypothesis testing, timebox, note todos in code	B/D/A	I	1,3,5,6,7,8,9	

Different perspectives: Appreciating a different perspective, coupled with associated relevant feedback, can help avoid biases such as Preconception (CB1), Fixation (CB3), and Superficial selection (CB9). Being exposed to different methods can help break developers out of cognitive "boot loops" by forcing them to reconsider, evaluate, and justify any subsequent action. For example, pair programming can help with Superficial selection (CB9) as the navigator can point out any errors in reasoning when programming.

Systematic approach: To avoid falling victim to biases or other errors, individuals should systematically approach the problem space and explore available solutions and tools. Such systematic review of different task parameters can help in avoiding biases such as Preconception (CB1), Memory (CB10), and Fixation (CB3) as developers will be both better aware of potential pitfalls, and also can consider alternate solutions ahead of time. As IP7 explained the practice of paying attention to documentation: [14:49]*"... [when choosing a tool] one of our criteria was [researching] how well is it documented? And I think it [good documentation] is very important."*

In addition to alternate solutions, systematic exploration helps developers keep the "big picture" in mind. In other words, it forces developers to more explicitly appreciate and acknowledge the larger goal, hopefully minimizing the likelihood that they will be distracted when *in situ*. This can prevent biases such as Ownership (CB2), by promoting the use of existing relevant code (that does not necessarily just belong to a single developer), which helps keep the larger code base backwards compatible.

RTFM: Consulting documentation before starting a task can avoid biases such as, Preconception (CB1), Memory (CB10), and Ownership (CB2), as developers can become aware of the multiple ways to problem solve and the pitfalls of each solution. For example, *Team diaries* and *playbooks* are journals where developers record guidelines for libraries and packages that specify how to use any code artifacts and avoid pitfalls.

Standardized, descriptive documentation of how to handle errors (or warnings) along with their severity levels can also help overcome biases such as Blissful ignorance (CB8) and Optimism (CB5) by helping developers locate faults more quickly.

Processes: Good software engineering practices such as designing and testing early and frequently, agile software development, etc. can help avoid biases in all categories to some extent. Developers can avoid biases such as Ownership (CB2) and Resort to Default (CB4) through coding standards and the use of standard libraries. This also helps developers to locate appropriate code to reuse.

Finally, effective problem-solving strategies can also help avoid biases such as Fixation (CB3), Convenience (CB6), and Subconscious Actions (CB7). For example, *convergent thinking exercises*—identifying a concrete solution to a problem—can help developers reach a (specific) solution quickly, whereas *divergent thinking exercises*—exploring multiple solutions to problems—can help developers identify an optimal solution from a set of alternatives. These prevent developers from fixating (CB3) on a single solution.

Tool wishlist. Participants felt that tool support to help overcome biases was lacking, and had difficulty naming any tools that they would use. They recommended the following tools that they wished existed to help deal with each of the listed biases:

Fixation (CB3): bias can be reduced by IDEs that track developer actions and detect situations where a developer is "fixated." It can then prompt different actions. IP12 explained, [12:44] *"...The IDE–if you change [code] and you always get the same error, it can say, hey, you have been do the same thing 5 times. But you always get the same error, maybe try something different?"*

Resort to default (CB4): It is a bias that developers will succumb to because it is a path of least resistance; as IP12 mentioned, "If there are default options, they'll just use it," and the way to overcome this problem would be via tool personalization and specification. He felt that tools' defaults should better match the current work context. For example, implementation of a high-level "intention" wizard that allows developers to "feed their intentions" into the wizard, which in turn then creates correct defaults and parameters relevant to the task.

Optimism (CB5): Tools that continuously run tests (and build scripts) in the background can counter this bias by identifying faulty changes that the developer might not have verified. IP12 recommended [12:44] *"[the tool] could figure out, 'hey! this is [code area] where I could run the tests' and it'd run the tests for you without you having to doing anything*.

However, he warned that such tools can become intrusive and distracting to the developer if they continuously notify developers of failing tests.

Convenience (CB6): bias can be prevented by a tool that can identify suboptimal code changes and recommend "clean" or "non-smelly" code. Not having "quick fix" changes can also help maintain backward compatibility and reduce technical debt. As IP1 explained, [14:36] *"some tools that could identify a quick fix ...And then point out some of the problems that this particular fix will cause*.

Subconscious actions (CB7): based on misleading and recurrent environmental cues can be prevented by annotating the severity of failures, exceptions, or results of flaky tests. IP12 mentioned annotating flaky tests, [42:45] *"updating the cues to say, well, it's not a red, it's a blood red! Because there is a test that we know shouldn't fail is failing. A test that has never failed in the past 20 builds, did now!"*

Blissful ignorance (CB8): can be avoided by tools that highlight a problem that appears similar to what the developer has experienced before and would otherwise ignore. Both IP15 and IP12 described a tool that allows developers to mark certain expected failures, such that the tool can notify them of other related failures.

Memory (CB10): bias can be avoided by a tool that automatically identifies deprecated methods and recommends the relevant updated API functions, instead of the function that the developer remembered. IP13 mentioned:[12:44] *"API code is evolving very frequently...and you don't know [the*

updated] methods…so one way to tell you like, hey, there's probably a better way of doing this."

6. DISCUSSION

Our results indicate that cognitive biases frequently disrupt development, and compromise developers' problem-solving abilities both in terms of task performance and time invested. Although developers currently deal with biases using a combination of standard and impromptu practices, there is a lack of tools that prevent or help developers recover from biases. Our findings have the following implications:

Implication for developers. Developers should be made aware that biases pose a significant threat to productive development, and perhaps are more pervasive than they realize. We synthesized a list of helpful practices (Table 6) that are expected to reduce the effect of cognitive biases. Some of these biases require an organizational level initiative. However, there are many practices that developers can individually employ on their own (for example, divergence thinking, defensive programming, and so forth). Interviewees often discussed that such practices have long-term benefits.

Implications for tool builders. Our interviews revealed that developers perceived a lack of tool support for dealing with biases. In Section 5.3, we identified various tool features that developers envisioned might help deal with biases. Further, as developers currently rely on a combination of standard and improvised practices to deal with biases, these practices need better tool support for effective implementation. Our results represent an initial starting point for tool builders to actualize tools that help prevent and deal with frequently demonstrated cognitive biases.

Limitations. Such as any field study, certain threats exist in our study that might challenge our findings. We describe some of these threats and steps taken to mitigate them.

Although our observational findings are derived from a small number of participants from a single software development company, the startup nature of the company ensure variation among the participants in terms of tasks and tools. Our primary units of analysis were the 2084 participants' actions (as opposed to the individual participants). To bolster these observations, we subsequently used a more diverse interview sample, which included participants from both large and small employers.

Observational studies are prone to confounds such as response bias, which can influence participant responses during our study. We mitigate this threat by having only one researcher directly observe a participant during our study, but supported behind the scenes by a second observer.

7. CONCLUSION

In this paper, through a field study of 10 developers, we investigated both how often cognitive biases occur in the workplace, and how these biases impact development. Our results indicate that cognitive biases frequently disrupt development, and compromise developers' problem-solving abilities such as exploration, sense-making, and contextual awareness. We compiled an initial set of practices and tools that developers currently use (or desire) to deal with cognitive biases.

The current findings provide a useful starting point for future investigations, and future efforts at developing a deeper understanding of cognitive biases will help developers and researchers to implement more effective preventive practices, and guide tool builders in creating curated support.

Acknowledgments

This work is partially supported by the National Science Foundation under Grant Nos. 2008089 and 1815486.

References

1. Calikli, G., Bener, A. 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 (PROMISE '10). Association for Computing Machinery, New York, NY, USA, 2010, Article 10, 1–11. DOI: https://doi.org/10.1145/1868328.1868344.
2. Calikli, G., Bener, A., Arslan, B. An analysis of the effects of company culture, education and experience on confirmation bias levels of software developers and testers. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—Volume 2 (ICSE '10)*. Association for Computing Machinery, New York, NY, USA, 2010, 187–190. DOI: https://doi.org/10.1145/1810295.1810326.
3. Chattopadhyay, S., Nelson, N., Gonzalez, Y.R., Leon, A.A., Pandita, R., Sarma, A. Latent patterns in activities: a field study of how developers manage context. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, 2019, 373–383. DOI: https://doi.org/10.1109/ICSE.2019.00051.
4. Cruz, S.S.J.O., da Silva, F.Q.B., Monteiro, C.V.F., Santos, P., Rossilei, I., dos Santos, M.T. Personality in software engineering: preliminary findings from a systematic literature review. *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, 2011, 1–10. DOI: 10.1049/ic.2011.0001
5. Easterbrook, S., Singer, J., Storey, M.A., Damian, D. Selecting empirical methods for software engineering research. In: F. Shull, J. Singer, D.I.K. Sjøberg, eds. Guide to Advanced Empirical Software Engineering. Springer, London, 2008. DOI: https://doi.org/10.1007/978-1-84800-044-5_11.
6. Jorgensen, M., Grimstad, S. Software development estimation biases: The role of interdependence. *IEEE Trans. Software Eng. 38*, 3 (2012), 677–693.
7. Mayer, R.E. *Thinking, Problem Solving, Cognition. A Series of Books in Psychology*. WH Freeman/Times Books/Henry Holt & Co, Worth Publishers, New York, 1992. ISBN-13: 9780716722151.
8. Miles, M.B., Huberman, A.M., Huberman, M.A., Huberman, M. *Qualitative Data Analysis: An Expanded Sourcebook*. SAGE, 1994.
9. Mohanani, R., Salman, I., Turhan, B., Rodríguez, P., Ralph, P. Cognitive biases in software engineering: A systematic mapping study. *IEEE Trans. Software Eng 46*, 12 (2020), 1318–1339. DOI: https://doi.org/10.1109/TSE.2018.2877759.
10. Pirolli, P. Computational models of information scent-following in a very large browsable text collection. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. Association for Computing Machinery, New York, NY, USA, 1997, 3–10. DOI: https://doi.org/10.1145/258549.258558.
11. Pirolli, P., Card, S. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In Volume 5 of *Proceedings of International Conference on Intelligence Analysis*. 2005, 2–4.
12. Runeson, P., Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Eng. 14*, 2 (2009), 131.
13. Sharpe, D. Your chi-square test is statistically significant: Now what? *Pract. Assess. Res. Eval. 20*, (01 2015), 1–10.
14. Sheskin, D.J. *Handbook of Parametric and Nonparametric Statistical Procedures: Third Edition* (3rd ed.). Chapman and Hall/CRC, New York, 2003. eBook ISBN 9780429186165. DOI: https://doi.org/10.1201/9781420036268.
15. Sillito, J., Murphy, G.C., De Volder, K. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '06/FSE-14)*. Association for Computing Machinery, New York, NY, USA, 2006, 23–34. DOI: https://doi.org/10.1145/1181775.1181779.
16. Silverman, B.G. Critiquing human judgment using knowledge-acquisition systems. *AI Mag. 11*, 3 (1990), 60.
17. Tversky, A., Kahneman, D. Judgment under uncertainty: Heuristics and biases. *Science 185*, 4157 (1974), 1124–1131.
18. Weinberg, G.M. *The Psychology of Computer Programming*, Volume 932633420. Van Nostrand Reinhold, New York, 1971.

Souti Chattopadhyay, Nicholas Nelson, Audrey Au, Natalia Morales, Christopher Sanchez, and Anita Sarma, Oregon State University, Corvallis, OR, USA.

Rahul Pandita, Phase Change Software, Golden, CO, USA.