Sorry to Bother You Again: Developer Recommendation Choice Architectures for Designing Effective Bots

Chris Brown, Chris Parnin
North Carolina State University
Raleigh, North Carolina, USA
{dcbrow10,cjparnin}@ncsu.edu

ABSTRACT

Software robots, or bots, are useful for automating a wide variety of programming and software development tasks. Despite the advantages of using bots throughout the software engineering process, research shows that developers often face challenges interacting with these systems. To improve automated developer recommendations from bots, this work introduces ***developer recommendation choice architectures***. Choice architecture is a behavioral science concept that suggests the presentation of options impacts the decisions humans make. To evaluate the impact of framing recommendations for software engineers, we examine the impact of one choice architecture, *actionability*, for improving the design of bot recommendations. We present the results of a preliminary study evaluating this choice architecture in a bot and provide implications for integrating choice architecture into the design of future software engineering bots.

KEYWORDS

software engineering, recommendations, developer behavior, choice architecture

ACM Reference Format:
Chris Brown, Chris Parnin. 2020. Sorry to Bother You Again: Developer Recommendation Choice Architectures for Designing Effective Bots. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3387940.3391506

1 INTRODUCTION

Bots are useful for automatically completing a wide variety of software engineering tasks to aid developers and support them in their work. For example, DependaBot is a system for automating security fixes on GitHub and Greenkeeper helps developers monitor npm dependencies for JavaScript packages. Furthermore, Wessel and colleagues show that bots are widely adopted in open source software (OSS) repositories and developers report finding bots beneficial for automating tasks such as explaining project guidelines, decreasing code review time, automating continuous integration, running tests and quality assurance, and more [30]. Additionally, Storey suggests bots can help developers become more efficient and effective in meeting their development goals [26].

Even though bots are useful for automating many different tasks and provide benefits to users, developers often face issues interacting with software bots designed to help them complete programming tasks. For instance, Wessel and colleagues also reported that software engineers face many challenges interacting with bots while contributing to and maintaining OSS projects including poor decision mechanisms, incomprehensible feedback, performing incorrect actions, and more [30]. Furthermore, Mirhosseini and colleagues discovered that developers had negative perceptions of automated pull requests and faced challenges convincing users to upgrade out-of-date dependencies with Greenkeeper and Travisbot [19]. These examples point to a need to improve the design of bots and better their interactions with human developers.

To identify a baseline design for automated bot recommendations to developers, our prior work introduced a naive approach static analysis tool recommendations [4]. We found that our bot recommendations were ineffective because our system violated social contexts within software engineering and interrupted the development workflow of programmers. To fix these problems in bot recommendations, we propose ***developer recommendation choice architectures***. Choice architecture refers to the way options are organized and presented to humans [28]. Johnson and colleagues introduced 11 tools to incorporate choice architecture into decisions for impacting user choices [16]. To help developers make better choices, we use these tools to present ***developer recommendation choice architectures*** for enhancing software engineering bots to improve developer behavior.

The goal of this work is to explore the impact of choice architecture on developer behavior adoption through automated recommendations. First, we introduce the ***developer recommendation choice architectures*** and provide examples of how effective choice architectures can improve decision-making. Then, we performed a preliminary evaluation incorporating one choice architecture, *actionability*, into automated notifications to collect feedback from developers. Finally, we introduce ***developer recommendation choice architectures*** to further improve the effectiveness of future bots by incorporating choice architectures into their design. Our results show that recommendations with actionability are significantly more effective and preferred by developers than those without. The primary contributions of this work are:

- introducing ***developer recommendation choice architectures*** that focus on the way recommendations are presented to software engineers to increase adoption, and
- a preliminary evaluation that explores integrating choice architecture into automated recommendations to developers.

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

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3391506

2 BACKGROUND

2.1 Choice Architecture

Brown and colleagues propose using *nudge theory*, or a behavioral science framework to improve human behavior and decision-making without banning alternatives or providing incentives [27], to improve human-bot interactions and the effectiveness of automated recommendations [4]. The driving force behind nudge theory is the ability to influence the context and environment surrounding human decision-making, or *choice architecture* [28]. Behavioral science research shows the way choices are framed and presented can have a major impact on human behavior, for example Wilson and colleagues show the arrangement of food options in cafeterias and grocery stores impacts the products people purchase and consume [31]. Furthermore, Johnson and colleagues introduce 11 practical tools for choice architecture, which we use in this work [16].

Thaler and Sunstein note that choice architecture "is pervasive and unavoidable" in human decision-making [27, p. 255]. This is also seen in software applications, where research has explored the impact of choice architecture on user interface design and software user behavior [23]. For example, Acquisti and colleagues explored creating choice architectures encouraging users to adopt better privacy and security practices online [1]. While prior work has studied choice architecture and nudge theory for software users, we study these concepts for software developers by applying choice architecture to recommendations from bots encouraging software engineers to adopt useful software engineering behaviors.

2.2 Automated Developer Recommendations

The role of bots in software engineering is an emerging research area.[1] Prior work has explored using bots to support *developer behaviors*, or useful practices and activities to help software engineers complete programming tasks, to users. For example, researchers have explored using bots to convince developers to adopt useful practices such as reducing developer effort to improve the code review process [2], upgrade outdated dependencies [19], update code documentation [22], and more. Prior work has also studied effective recommendations to developers. Research shows *peer interactions*, or face-to-face interactions with colleagues during work activities, are effective for tool discovery [20] and code comprehension [17], but they also occur infrequently in industry [21]. Meanwhile, Fischer and colleagues propose *active help systems* to automatically aid users are more useful than passive approaches [10].

For software engineering recommender bots, our prior work introduced the naive *telemarketer design*, a baseline approach that makes generic and static recommendations to developers similar to a telemarketer, and found that developers disapproved of this design in `tool-recommender-bot` because it lacks social context and interrupted developer workflow [4]. Additionally, Cerezo and colleagues developed an expert recommender chatbot and to examine interactions with humans [5], while Beschastnikh and colleagues propose using bots to increase adoption of software engineering research in industry [3]. This work seeks to improve developer behavior by incorporating choice architecture into bot recommendations to encourage adoption of useful tools and practices.

[1] http://botse.org/

3 *DEVELOPER RECOMMENDATION CHOICE ARCHITECTURES*

To further improve software engineering bots, we introduce ***developer recommendation choice architectures*** to design effective bots, motivated by software engineering literature and tools for choice architecture. Johnson suggests these 11 tools are useful for preventing decision-making problems by structuring decisions and describing options [16]. Table 1 presents how our developer choice architectures correlate with the tools available for choice architects to improve decision-making. We suggest software engineering researchers and bot designers are *choice architects*, or "anyone who presents people with choices", by creating automated notifications that developers need to make decisions on. To improve the decision-making process for automated developer recommendations, we propose bot makers emphasize *actionability*, *feedback*, and *locality*.

3.1 Actionability

Actionability refers to the ease with which users can act on recommendations. Nudge theory suggests "many people will take whatever option requires the least effort, or the path of least resistance" [27, p. 85]. Johnson provides several tools for improving actionability, including *technology and decision aids* and *use defaults*. Johnson and colleagues note "the default option will be chosen more often than if another option is designated the default" and that "technology-based decision aids could be designed to steer consumers towards choosing products, services, or activities that are individually and/or socially desirable" [16, p. 488,491]. For example, Madrian and Shea explored automatically enrolling employees in 401k plans. By changing the default behavior to having users opt-out of retirement plans instead of making them opt in, they found that this improved money saving behaviors and increased new employee enrollment in 401k plans by 98% within 36 months [18].

Actionability has also shown to be effective in software engineering research. For example, Evans and colleagues show that configuring security checks by default in configurations for static analysis tools prevents vulnerabilities in the code [9]. Furthermore, prior work by Heckman and colleagues explored actionable alert identification techniques (AAITs) for static analysis tool notifications to help software engineers identify and fix defects in code earlier [14]. To design effective bots, we suggest making actionable recommendations by making targets behaviors default for decision-making of developers.

3.2 Feedback

Feedback focuses on the information provided to decision-makers. Behavioral scientists Sunstein and Thaler suggest "the best way to help Humans improve their performance is to provide feedback". Furthermore, they also posit "choices can be improved with better and simpler information" [27, p. 92,204]. Johnson and colleagues provide many different examples of improving information presented and the way it's communicated to users. For example, providing *customized information* on food calories and daily caloric intake feedback encouraged consumers to purchase healthier meals at fast food restaurants [32]. Additionally, changing gas and CO_2 emissions from 100 to 10,000 miles *translates and rescales for better evaluability* to impact car purchases [16].

Table 1: Map of Developer Recommendation Choice Architectures (DRCA) and Tools for Choice Architecture

DRCA	Tool [16]	Definition
Actionability	Technology and decision aids	Introducing technology to aid decision makers in choice tasks
	Use defaults	The way decision makers initially encounter choice tasks
Feedback	Reduce number of alternatives	Limiting the number of choice options presented to decision makers
	Focus on satisficing	Helping users consider outcomes that lead to higher choice satisfaction
	Attribute parsimony and labeling	Limiting the number of characteristics presented with options
	Translate and rescale for better evaluability	Presenting attributes to increase impact and clarity
	Customized information	Personalization to account for individual differences between decision-makers
	Focus on experience	Considering the background and knowledge of decision-makers
Locality	Limited time windows	Providing time restrictions for users to make decisions
	Partitioning of options	Groups or categories of options or attributes
	Decision staging	Dividing decisions into multiple stages

Software engineering researchers also show that feedback to developers is important. For instance, Johnson and colleagues discovered that the primary reasons developers reported for not using static analysis tools in their work is poor result understandability, customizability, and tool output [15]. Additionally, Chmiel and colleagues *focus on experience* to study information needs for novices and experts during debugging tasks [6]. Furthermore, research Cerezo and colleagues suggest implementing *user-driven communication* to improve chatbot recommender systems instead of single purpose bot-driven communication techniques [5]. To improve automated developer recommendations, we propose providing useful information and feedback to users.

3.3 Locality

Locality involves the setting surrounding the context of developer recommendations. To further define this, we divide it into *spatial* and *temporal* locality.

3.3.1 Spatial Locality. Spatial locality refers to the location where recommendations are viewed as well as how options are arranged. Johnson notes the structure of decision tasks not only impacts decisions, but also how users explore the option space and search for information, which "can have a dramatic impact on choice behavior" [16, p. 494]. For example, research shows that changing the location of vegetables, fruits, etc. in a high school cafeteria increased the purchase and consumption of healthier foods by students [13] and *partitioning options* from a menu into separate categories for healthy and unhealthy options encourages consumers to choose more healthy food and less unhealthy food [32].

Software engineering research also shows that the location of notifications matters to developers. For example, Smith and colleagues implemented Flower, an *in situ* code navigation tool within the code, and found that this increased efficiency and received positive responses from developers [24]. Furthermore, Viriyakattiyaporn and colleagues found that several users ignored recommendations from Spyglass, a code navigation plugin, because they were unaware of the location of notifications from the tool within the Eclipse IDE [29]. To improve the effectiveness of software engineering bots, we propose presenting recommendations in familiar and convenient locations for developers.

3.3.2 Temporal Locality. Temporal locality references the time when recommendations are made. Johnson notes timing impacts decisions, stating "the intertemporal structure of a task has important implications for both the decision-maker and the choice architect...which affects choice tasks " [16, p. 492]. For example, introducing *time-limited windows* for yearly fertilizer discounts encouraged farmers in Kenya to make purchases and improved the harvest of crops [8]. Additionally, timing also refers to when choices take place after decisions. For example, Johnson also notes "in general, tools that translate aspects of the choice into immediate salient outcomes are more successful" [16, p. 493]. For example, Soman and colleagues show that consumers view decisions with distant impacts differently than those with proximal outcomes [25].

Similarly in software engineering, prior work suggests timing is important for notifications. For instance, Distefano and colleagues examined configuring static analysis tools to run at *diff time*, or a limited time window for patches submitted by developers to review before merging into the code base, and found that this increased the fix rate of reported bugs up to 70% compared to nearly 0% for times outside the development workflow, such as assigning bug lists to developers from overnight builds [7]. Furthermore, our prior work shows that developers avoided recommendations from `tool-recommender-bot` because they made untimely suggestions interrupting their workflow by breaking builds and adding to the workload of programmers [4]. For improving bot recommendations, we propose automating suggestions to programmers in time windows within their workflow.

4 PRELIMINARY EVALUATION

To explore how integrating choice architecture into developer recommendations impacts the behavior of software engineers, we start by conducting a preliminary evaluation examining the impact of one of our ***developer recommendation choice architectures***: *actionability*. We evaluated actionability in bots by surveying professional software engineers to provide feedback recommendations from two systems on GitHub, pull request comments and suggested changes. These systems have make similar recommendations to programmers, but only differ on the actionability for developers to apply suggestions to their code. Here, we explain our methodology and early results from this introductory evaluation.

```
9   if status is True:
10  -   print 'passed'
11  +   print('passed')
```

Listing 1: Example of `PEP 3105` violation and fix

4.1 Methodology

4.1.1 Choice Task. In this evaluation, sample recommendations recommended fixing a `PEP 3105` Python style warning, because `print` statements are replaced with a `print()` function in Python 3 [11]. Listing 1 shows a sample violation and fix for this simple warning. Additionally, as of January 1, 2020 Python is officially no longer supporting Python 2 [12]. Our sample recommendations encourage developers to fix a `PEP 3105` warning and upgrade to Python 3.

4.1.2 Study Recommendations. To measure the effectiveness of automated recommendations with and without actionability, we presented developers with recommendations from two systems for providing feedback on pull requests. In our study, both systems provide the same feedback, a proposed fix for the `PEP 3105` warning in Listing 1, and encouraged users to upgrade to Python 3. Additionally, both have the same locality- placed on line 10 in the example above and would be received during reviews on open PRs. These systems differ, however, on the actionability of recommendations. A pull request comment is not actionable and requires developers to re-submit a pull request to make the change. However, a suggested change is highly actionable because it allows developers to immediately commit suggestions to their code. By using this technology aid and it's default action to commit proposed code changes to pull requests, we aim to show that developers prefer actionable recommendations to easily integrate suggestions into their workflow.

4.1.3 Data Collection. To discover the impact of our actionability choice architecture on developer recommendations, we surveyed software developers to provide their preference for receiving recommendations via pull request review comment or suggested change. After presenting example notifications with each option, we asked participants to select which recommendation they preferred and why. Additionally, we encouraged developers to provide general feedback on designing effective automated recommendations from bots. Overall, 15 developers completed the survey with an average of 7.3 years of programming experience.

4.2 Results

In our survey responses, all 15 developers preferred the actionable recommendation from suggested changes over the non-actionable pull request review comment. Table 2 presents the results from our survey. This shows that developers were significantly more likely to adopt recommendations from actionable systems that make it easier to apply suggestions from bots. Developers also provided feedback praising the actionable recommendation. The *default* behavior for suggested changes is to commit recommendations, which most participants mentioned made it more effective. For example, P2 commented that the suggested changes recommendation would "provide an actionable short cut" and P8 liked that it "lets you automatically merge it". Even though both recommendations contained the exact same feedback information, approximately half of participants ($n = 7$) replied that the actionable recommendation was easier to understand. For instance P1 noted it's "more specific and helps the user understand better". This could be due to the *technology aids* such coloring, highlighting, and side-by-side comparisons in the suggested changes feature.

	n	**Percent**
comment	0	0%
suggestion	15	100%

Table 2: Actionable Recommendation Survey Results

5 IMPLICATIONS AND FUTURE WORK

Our results suggest that integrating concepts from behavioral science and choice architecture can improve how software engineers perceive bot recommendations. We found that developers significantly preferred actionable recommendations as opposed to non-actionable ones. Based on this, we argue that software engineering bot developers and researchers are choice architects, presenting developers with automated recommendations and choices during programming activities. To improve the behavior and decision-making of software engineers, we propose integrating ***developer recommendation choice architectures*** into the design of bot systems can help make more effective notifications to improve developer recommendations and decision-making.

As choice architects, researchers and bot developers can improve the effectiveness of software engineering bots by incorporating ***developer recommendation choice architectures*** into automated recommendation from systems. For future work, we plan to explore additional ways to incorporate choice architecture and behavioral science into the design of software engineering bots. We plan to implement and evaluate bots that also vary *feedback* and *locality* to determine their impact on recommendations to developers. Furthermore, we aim to discover new choice architectures to continue to improve software engineering bots encouraging developers to adopt useful practices and tools. Future work can also explore the impact of ***developer recommendation choice architectures*** on other metrics, such as the amount of time to adoption.

6 CONCLUSION

We present ***developer recommendation choice architectures*** to improve the effectiveness of automated recommendations to software engineers. By incorporating *actionability*, *feedback*, and spatial and temporal *locality* into automated bot notifications, we believe these can improve the way decisions are presented to developers and encourage adoption of useful tools and practices. To discover the impact of integrating these design principles into automated recommendations, we compared the effectiveness of two systems to suggest fixing Python `PEP 3105` warnings that differed in actionability, or the ease with which developers could apply recommendations to their code.

REFERENCES

[1] Alessandro Acquisti, Idris Adjerid, Rebecca Balebako, Laura Brandimarte, Lorrie Faith Cranor, Saranga Komanduri, Pedro Giovanni Leon, Norman Sadeh, Florian Schaub, Manya Sleeper, et al. 2017. Nudges for privacy and security: Understanding and assisting users' choices online. *ACM Computing Surveys (CSUR)* 50, 3 (2017), 44.

[2] V. Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 931–940. https://doi.org/10.1109/ICSE.2013.6606642

[3] I. Beschastnikh, M. F. Lungu, and Y. Zhuang. 2017. Accelerating Software Engineering Research Adoption with Analysis Bots. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*. IEEE, 35–38. https://doi.org/10.1109/ICSE-NIER.2017.17

[4] C. Brown and C. Parnin. 2019. Sorry to Bother You: Designing Bots for Effective Recommendations. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 54–58. https://doi.org/10.1109/BotSE.2019.00021

[5] J. Cerezo, J. Kubelka, R. Robbes, and A. Bergel. 2019. Building an Expert Recommender Chatbot. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 59–63. https://doi.org/10.1109/BotSE.2019.00022

[6] Ryan Chmiel and Michael C Loui. 2004. Debugging: from novice to expert. *ACM SIGCSE Bulletin* 36, 1 (2004), 17–21.

[7] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling Static Analyses at Facebook. *Commun. ACM* 62, 8 (July 2019), 62–70. https://doi.org/10.1145/3338112

[8] Esther Duflo, Michael Kremer, and Jonathan Robinson. 2011. Nudging farmers to use fertilizer: Theory and experimental evidence from Kenya. *American economic review* 101, 6 (2011), 2350–90.

[9] David Evans and David Larochelle. 2002. Improving security using extensible lightweight static analysis. *IEEE software* 19, 1 (2002), 42–51.

[10] Gerhard Fischer, Andreas Lemke, and Thomas Schwab. 1984. *Active help systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 115–131. https://doi.org/10.1007/3-540-13394-1_10

[11] Python Software Foundation. 2006. PEP 3105 – Make print a function. https://www.python.org/dev/peps/pep-3105/.

[12] Python Software Foundation. 2019. Sunsetting Python 2. https://www.python.org/doc/sunset-python-2/.

[13] Andrew S. Hanks, David R. Just, Laura E. Smith, and Brian Wansink. 2012. Healthy convenience: nudging students toward healthier choices in the lunchroom. *Journal of Public Health* 34, 3 (01 2012), 370–376. https://doi.org/10.1093/pubmed/fds003 arXiv:http://oup.prod.sis.lan/jpubhealth/article-pdf/34/3/370/12782601/fds003.pdf

[14] Sarah Heckman and Laurie Williams. 2011. A systematic literature review of actionable alert identification techniques for automated static code analysis. *Information and Software Technology* 53, 4 (2011), 363 – 387. https://doi.org/10.1016/j.infsof.2010.12.007 Special section: Software Engineering track of the 24th Annual Symposium on Applied Computing.

[15] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 672–681. https://doi.org/10.1109/ICSE.2013.6606613

[16] Eric J Johnson, Suzanne B Shu, Benedict GC Dellaert, Craig Fox, Daniel G Goldstein, Gerald Häubl, Richard P Larrick, John W Payne, Ellen Peters, David Schkade, et al. 2012. Beyond nudges: Tools of a choice architecture. *Marketing Letters* 23, 2 (2012), 487–504.

[17] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the Comprehension of Program Comprehension. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 31 (Sept. 2014), 37 pages. https://doi.org/10.1145/2622669

[18] Brigitte C Madrian and Dennis F Shea. 2001. The power of suggestion: Inertia in 401 (k) participation and savings behavior. *The Quarterly journal of economics* 116, 4 (2001), 1149–1187.

[19] S. Mirhosseini and C. Parnin. 2017. Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 84–94. https://doi.org/10.1109/ASE.2017.8115621

[20] Emerson Murphy-Hill, Da Young Lee, Gail C. Murphy, and Joanna McGrenere. 2015. How Do Users Discover New Tools in Software Development and Beyond? *Computer Supported Cooperative Work (CSCW)* 24, 5 (2015), 389–422. https://doi.org/10.1007/s10606-015-9230-9

[21] Emerson Murphy-Hill and Gail C. Murphy. 2011. Peer Interaction Effectively, Yet Infrequently, Enables Programmers to Discover New Tools. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work (CSCW '11)*. ACM, New York, NY, USA, 405–414. https://doi.org/10.1145/1958824.1958888

[22] S. Rebai, O. Ben Sghaier, V. Alizadeh, M. Kessentini, and M. Chater. 2019. Interactive Refactoring Documentation Bot. In *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 152–162. https://doi.org/10.1109/SCAM.2019.00026

[23] Christoph Schneider, Markus Weinmann, and Jan Vom Brocke. 2018. Digital nudging: guiding online user choices through interface design. *Commun. ACM* 61, 7 (2018), 67–73.

[24] J. Smith, C. Brown, and E. Murphy-Hill. 2017. Flower: Navigating program flow in the IDE. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 19–23. https://doi.org/10.1109/VLHCC.2017.8103445

[25] Dilip Soman, George Ainslie, Shane Frederick, Xiuping Li, John Lynch, Page Moreau, Andrew Mitchell, Daniel Read, Alan Sawyer, Yaacov Trope, et al. 2005. The psychology of intertemporal discounting: Why are distant events valued differently from proximal ones? *Marketing Letters* 16, 3-4 (2005), 347–360.

[26] Margaret-Anne Storey and Alexey Zagalsky. 2016. Disrupting Developer Productivity One Bot at a Time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 928–931. https://doi.org/10.1145/2950290.2983989

[27] Richard H Thaler and Cass R Sunstein. 2009. *Nudge: Improving decisions about health, wealth, and happiness*. Penguin, New York, NY, USA.

[28] Richard H Thaler, Cass R Sunstein, and John P Balz. 2014. *Choice architecture*. Princeton University Press, Chapter 25.

[29] P. Viriyakattiyaporn and G. C. Murphy. 2009. Challenges in the user interface design of an IDE tool recommender. In *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. IEEE, 104–107. https://doi.org/10.1109/CHASE.2009.5071421

[30] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. 2018. The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 182 (Nov. 2018), 19 pages. https://doi.org/10.1145/3274451

[31] Amy L Wilson, Elizabeth Buckley, Jonathan D Buckley, and Svetlana Bogomolova. 2016. Nudging healthier food and beverage choices through salience and priming. Evidence from a systematic review. *Food Quality and Preference* 51 (2016), 47–64.

[32] Jessica Wisdom, Julie S. Downs, and George Loewenstein. 2010. Promoting Healthy Choices: Information versus Convenience. *American Economic Journal: Applied Economics* 2, 2 (April 2010), 164–78. https://doi.org/10.1257/app.2.2.164