Dark Patterns for Influencing Developer Behavior

CHRIS BROWN, North Carolina State University, USA
CHRIS PARNIN, North Carolina State University, USA

The behavior of software engineers has a major impact on the technology we use everyday. Unfortunately, research shows developers often demonstrate bad behaviors and make poor decisions in their work. Existing literature has explored using automated tools to recommend useful behaviors to programmers. However, studies show these system-to-user suggestions are often ineffective while user-to-user suggestions between peers are the most effective recommendation approach. To improve automated recommendations for developer behaviors, this work delves into two interdisciplinary methods to influence human behavior: *nudge theory* and *dark patterns*. We present a comparison of these two approaches for impacting the behavior and decision-making of humans, and offer prototype designs for dark pattern recommender bots to generate suggestions to improve the choices and behavior of software developers.

Additional Key Words and Phrases: dark patterns, nudge theory, developer behavior, automated recommendations, recommender bots

ACM Reference Format:
Chris Brown and Chris Parnin. 2021. Dark Patterns for Influencing Developer Behavior. 1, 1 (March 2021), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Decision-making is a vital part of software engineering, regarded as "the most undervalued" and "most important" skill in software development.[1] However, developers frequently make bad decisions in their work. For instance, despite scientific evidence of their benefit, research shows developers often avoid using static analysis tools to automatically find defects in code [12] and prevent security vulnerabilities [25]. Ultimately, these poor decisions by software engineers are costly. Studies show debugging, the process of finding and fixing errors, consumes the majority of development costs and developers' time [18] while software failures impact billions of users and cost trillions of dollars each year.[2]

Software engineering literature has explored various ways to improve the decision-making of developers. For example, the Association for Computing Machinery (ACM) published a Code of Ethics to guide the decisions of programmers while developing and maintaining software applications [8]. However, research shows developers often ignore these guidelines [14]. For instance, even though the ACM Code of Ethics encourages computing professionals to be "honest and trustworthy"[3] and "ensure that the public good is the central concern",[4] developers still make unethical decisions such as implementing *dark patterns*, or deceptive user interfaces that often trick users to modify their behavior online.

[1] https://hackernoon.com/decision-making-the-most-undervalued-skill-in-software-engineering-f9b8e5835ca6
[2] https://www.tricentis.com/news/tricentis-software-fail-watch-finds-3-6-billion-people-affected-and-1-7-trillion-revenue-lost-by-software-failures-last-year/
[3] https://www.acm.org/code-of-ethics#h-1.3-be-honest-and-trustworthy
[4] https://www.acm.org/code-of-ethics#h-3.1-ensure-that-the-public-good-is-the-central-concern-during-all-professional-computing-work

Authors' addresses: Chris Brown, dcbrow10@ncsu.edu, North Carolina State University, Department of Computer Science, Raleigh, North Carolina, USA, 27605; Chris Parnin, cjparnin@ncsu.edu, North Carolina State University, Department of Computer Science, Raleigh, North Carolina, USA, 27695.

Software engineering researchers have also explored automated approaches to improve the choices and behavior of programmers [20]. Despite the value of bots in software development [23], studies suggest recommendations from these systems are often ineffective for influencing developer behavior [3]. Moreover, research shows peer interactions, or recommendations between colleagues during work activities, are the most effective method for making recommendations to developers compared to technical approaches [15]. However, opportunities for these in-person suggestions between developers in industry are infrequent and declining [16], motivating the need for automated techniques.

To improve the efficacy of automated recommendations, our prior work proposes incorporating concepts from *nudge theory* into bots [5]. Nudge theory is a behavioral science framework for improving human behavior by influencing decisions without 1) providing incentives or 2) banning alternative choices [21]. Dark patterns are another form of indirect influence to alter human behavior, and in this work we propose using this concept to improve the decision-making of programmers while they develop and maintain software applications. The contributions of this work are a comparison of nudge theory and dark patterns as mechanisms for modifying human behavior and examples of systems using dark pattern strategies to convince developers to adopt better behaviors.

2 BACKGROUND

2.1 Nudge Theory

Behavioral science research suggests nudges are effective for improving human decision-making [21]. For example, studies show placing healthy options near the front of a cafeteria encourages students to purchase and consume healthier options [11]. Furthermore, *digital nudging* is the process of using technology to nudge humans toward better behaviors in digital choice environments [22]. For example, the Square mobile payment app changed the default user action to tip merchants which led to a 133% increase in tips.[5] This intervention is a nudge because it does not reward users for choosing to tip nor force gratuity as the only option. Prior work suggests digital nudges are beneficial for encouraging humans to adopt better privacy behaviors online [1] and reduce social media usage [19].

Nudges are impactful for improving human behavior because of their ability to influence the context and environment surrounding decisions, or *choice architecture* [21]. To encourage software engineers to adopt better behaviors, we introduced *developer recommendation choice architectures* as a novel framework that incorporates concepts from nudge theory to improve the effectiveness of automated recommendations [5]. By incorporating ***actionability***, ***feedback***, and ***locality*** into the design of bots, we argue these systems can nudge developers to adopt useful tools and practices. For instance, we found developers prefer actionable static analysis tool recommendations over static approaches [4].

2.2 Dark Patterns

Dark patterns are user interface designs that deceptively influence the decision-making and behavior of users online.[6] User experience (UX) and human factors researchers show dark patterns can influence the choices of humans. For instance, studies show websites often incorporate misleading and dishonest user interface designs that impact users' online shopping behaviors [13] and their divulgence of private data [6]. Prior work suggests dark patterns evolved from explorations into nudge theory [17]. Similar to digital nudges, dark patterns rely on covert and subliminal techniques to cognitively influence the decision-making of users in digital choice environments. However, while the intention of nudges is to improve human decision-making and encourage the adoption of beneficial target behaviors [21], dark patterns often work against user interests and trick them into adopting behaviors favorable for the designers.[5]

[5] https://www.fastcompany.com/3022182/how-square-registers-ui-guilts-you-into-leaving-tips
[6] https://darkpatterns.org/

Dark patterns patterns are primarily viewed as unethical and widely discouraged due to their deceptive nature [9], however researchers have considered opportunities where using technology to trick users is beneficial [10]. For example, Fogg and colleagues present design patterns for creating technology to persuade users to adopt desired behaviors [7]. In this work, we aim to explore the following question: can dark patterns be used to trick software engineers into adopting better programming behaviors? While most research examining digital nudges and dark patterns investigates their impact on the decision-making of software users, this work seeks to explore how these concepts can influence the choices and behavior of developers who build and maintain the software we use.

3 DARK PATTERN RECOMMENDATIONS

To improve the decision-making of developers, we present examples of automated systems incorporating dark patterns to recommend beneficial software engineering behaviors to developers. In this case, the behavior we focus on is static analysis tool adoption. Research shows utilizing static analysis tools provides a variety of benefits to development teams, such as preventing errors and improving code quality [2]. However, studies also show that software engineers rarely use these automated code-checking systems in practice [12]. The design of these sample recommendations incorporate five strategies for dark patterns introduced in prior work by Gray and colleagues: *nagging*, *obstruction*, *sneaking*, *interface interference*, and *forced action* [10]. Here, we briefly describe sample automated recommendations with each technique to improve static analysis tool adoption among software engineers.

Nagging. Nagging refers to repeated intrusions to influence user behavior while completing a task. For instance, the Microsoft Clippy help system was designed to guide users, but they found it's interruptions frustrating and irritating [24]. To encourage developers to adopt static analysis tools in their work, one approach is to consistently nag programmers to incorporate code-checking systems into their workflow. For example, Figure 1a presents a sample bot that generates nagging automated GitHub issues to recommend static analysis tools to developers on coding projects.

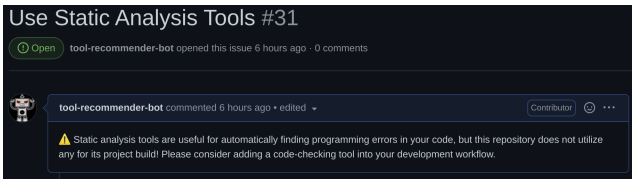
Obstruction. Obstruction concerns design decisions that create barriers inhibiting users from completing tasks in order to force certain behaviors and actions. For instance, Clubhouse blocks users from adding contacts unless they provide access to their data from Twitter.[7] An example of this strategy for increasing the adoption of static analysis tools is coding editors or code hosting platforms preventing users from merging contributions into repositories unless the changes to the program have been analyzed by a code-checking tool (see Figure 1b).

Sneaking. Sneaking refers to hiding or disguising information relevant to users' decision-making processes. An example of this strategy can be found in online ticket purchasing websites such as TicketMaster, which frequently have hidden service fees and convenience costs not revealed to customers until checkout.[8] Likewise, covert approaches can be used to increase code quality by automatically running static analysis tools on source code in integrated development environments (IDEs) while developers are completing programming tasks without their knowledge or permission.

Interface Interference. Interface interference concerns manipulating user interface designs to trick users into adopting certain behaviors. For example, TurboTax modified their website to hide their free tax filing program to coerce users into using their paid platform.[9] Examples of interface interference to convince developers to increase static analysis tool usage involve changing programming environments until a development tool has been run on the code (i.e., graying out options to disallow committing code in IDEs until a code-checking tool is run in Figure 1c).

[7] https://medium.com/privacy-technology/when-fomo-trumps-privacy-the-clubhouse-edition-82526c6cd702
[8] https://www.huffpost.com/entry/concert-ticket-hidden-fees_l_5dfd1021e4b05b08bab527ef
[9] https://www.propublica.org/article/turbotax-just-tricked-you-into-paying-to-file-your-taxes

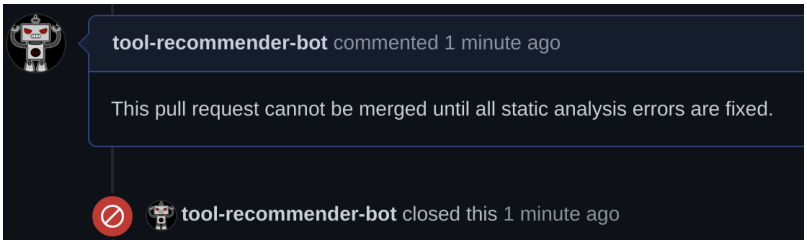
Forced Action. Forced action involves demanding users perform specific activities in order to continue or complete a task. An example of this can be seen in WhatsApp, which requires users to share their data with Facebook in order to use the app.[10] While a property of nudges is that they cannot force choosers to select certain behaviors, dark patterns can force humans to adopt specific behaviors, such as requiring programmers to use static analysis tools. For example, Figure 1d depicts a bot that prevents pull requests from being merged until all reported static analysis bugs are fixed.



(a) Nagging

(b) Obstruction

(c) Interface Interference



(d) Forced Action

Fig. 1. Dark Pattern Recommendation Examples

4 IMPLICATIONS AND FUTURE WORK

This work explores using dark patterns to design automated recommendations for improving the behavior of software engineers. As our society becomes increasingly dependent upon technology, the need for developers to adopt beneficial programming behaviors and make better decisions also grows. One example of a beneficial behavior developers often avoid is incorporating static analysis tools into their workflow, despite the fact these systems provide many benefits to development teams by automatically finding and reporting errors in code. To increase the adoption of these systems, this work proposes several designs for developer recommender bots incorporating dark patterns.

Future directions of this research involve implementing the dark pattern recommendations in automated systems and evaluating their impact on developer behavior. We hypothesize certain dark pattern strategies will be more effective than others. For instance, nagging would most likely be unsuccessful because prior work shows interrupting developers' workflow leads to ineffective recommendations [3]. Additionally, we aim to create bots to suggest good programming behaviors beyond static analysis tool adoption, such as ethical programming decisions. Furthermore, future work can explore applying concepts from other disciplines to improve the behavior and decision-making of developers. In this work, we attempt to portray dark patterns derived from UX research as a design mechanism for improving the effectiveness of automated systems to recommend useful development tools and practices to software engineers.

[10] https://bgr.com/2021/01/07/whatsapp-privacy-policy-change-data-sharing-facebook/

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] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix. 2008. Using Static Analysis to Find Bugs. *IEEE Software* 25, 5 (2008), 22–29. https://doi.org/10.1109/MS.2008.130

[3] Chris Brown and Chris Parnin. 2019. Sorry to bother you: designing bots for effective recommendations. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*. IEEE Press, 54–58.

[4] Chris Brown and Chris Parnin. 2020. Comparing Different Developer Behavior Recommendation Styles. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 78–85. https://doi.org/10.1145/3387940.3391481

[5] Chris Brown and Chris Parnin. 2020. Sorry to Bother You Again: Developer Recommendation Choice Architectures for Designing Effective Bots. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 56–60. https://doi.org/10.1145/3387940.3391506

[6] Christoph Bösch, Benjamin Erb, Frank Kargl, Henning Kopp, and Stefan Pfattheicher. 01 Oct. 2016. Tales from the Dark Side: Privacy Dark Strategies and Privacy Dark Patterns. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (01 Oct. 2016), 237 – 254. https://doi.org/10.1515/popets-2016-0038

[7] BJ Fogg. 2009. Creating Persuasive Technologies: An Eight-step Design Process. In *Proceedings of the 4th International Conference on Persuasive Technology* (Claremont, California, USA) *(Persuasive '09)*. ACM, New York, NY, USA, Article 44, 6 pages. https://doi.org/10.1145/1541948.1542005

[8] Association for Computing Machinery. 2018. ACM Code of Ethics and Professional Conduct. https://www.acm.org/code-of-ethics

[9] Colin M. Gray, Shruthi Sai Chivukula, and Ahreum Lee. 2020. *What Kind of Work Do "Asshole Designers" Create? Describing Properties of Ethical Concern on Reddit.* Association for Computing Machinery, New York, NY, USA, 61–73. https://doi.org/10.1145/3357236.3395486

[10] Colin M. Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin L. Toombs. 2018. The Dark (Patterns) Side of UX Design. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3173574.3174108

[11] 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

[12] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. 2. Conference. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 672–681. https://doi.org/10.1109/ICSE.2013.6606613

[13] Arunesh Mathur, Gunes Acar, Michael J. Friedman, Elena Lucherini, Jonathan Mayer, Marshini Chetty, and Arvind Narayanan. 2019. Dark Patterns at Scale: Findings from a Crawl of 11K Shopping Websites. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 81 (Nov. 2019), 32 pages. https://doi.org/10.1145/3359183

[14] Andrew McNamara, Justin Smith, and Emerson Murphy-Hill. 2018. Does ACM's Code of Ethics Change Ethical Decision Making in Software Development?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 729–733. https://doi.org/10.1145/3236024.3264833

[15] 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

[16] 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* (Hangzhou, China) *(CSCW '11)*. ACM, New York, NY, USA, 405–414. https://doi.org/10.1145/1958824.1958888

[17] Arvind Narayanan, Arunesh Mathur, Marshini Chetty, and Mihir Kshirsagar. 2020. Dark Patterns: Past, Present, and Future. *Commun. ACM* 63, 9 (Aug. 2020), 42–47. https://doi.org/10.1145/3397884

[18] National Institute of Standards and Technology. 2002. The economic impacts of inadequate infrastructure for software testing. *U.S. Department of Commerce Technology Administration* (2002).

[19] Aditya Kumar Purohit, Louis Barclay, and Adrian Holzer. 2020. Designing for Digital Detox: Making Social Media Less Addictive with Digital Nudges. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–9.

[20] Martin Robillard, Robert Walker, and Thomas Zimmermann. 2010. Recommendation systems for software engineering. *IEEE software* 27, 4 (2010), 80–86.

[21] Richard H Thaler and Cass R Sunstein. 2009. *Nudge: Improving decisions about health, wealth, and happiness*. Penguin.

[22] Markus Weinmann, Christoph Schneider, and Jan vom Brocke. 2016. Digital nudging. *Business & Information Systems Engineering* 58, 6 (2016), 433–436.

[23] 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. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 182.

[24] Brian Whitworth. 2005. Polite computing. *Behaviour & Information Technology* 24, 5 (2005), 353–363. https://doi.org/10.1080/01449290512331333700 arXiv:http://dx.doi.org/10.1080/01449290512331333700

[25] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing* (Baltimore, Maryland, USA) *(CSCW '14)*. ACM, New York, NY, USA, 1095–1106. https://doi.org/10.1145/2531602.2531722