RESET

RECHERCHES EN SCIENCES SOCIALES SUR INTERNET

SOCIAL SCIENCE RESEARCH ON THE INTERNET

reset@openedition.org
http://reset.revues.org
ISSN 4939–0247

ARTICLE

Discursive Strategies in Style Guides Negotiation on GitHub

PIERRE DEPAZ - PIERRE.DEPAZ@SORBONNE-NOUVELLE.FR

This article examines the discursive strategies at play in style guide negotiation on GitHub. Looking at popular guides for the JavaScript language, we highlight how source code, executable code, networks of communities and platform affordances are used as arguments of their own in the adoption and modification of theoretically immutable documents. Additionally, we show how programmers display multifaceted practices in a social context of work.

Keywords: Style guides, discursive negotiations, sociolinguistics, computer-supported cooperative work

1. Introduction

Written and published in 1983 on a Usenet board, *The story of Mel, the Real Programmer* recounts the tale of Mel Kaye, an individual who wrote software on the 1959 ACT-1 compiler and has become a recurring reference in programmer's lore (Nather, 2003). The story focuses on Kaye's ability to write both excellently efficient and completely inscrutable code. Code which only its writer can read, considered as model programming work and informing ideals of programmers, slowly began to phase out with commercial software.

The evolution, from the individual programmer implementing *ad hoc* and personal solutions to a group of programmers coordinating across time to build and maintain large, distributed pieces of software, brought the necessity to harmonize and standardize how code is written (van den Boogard, 2008). In response, style guides started to be published to normalize the visual aspect of source code, and became a recurring topic in both software development and computer science research (Kerninghan & Plauger, 1974).

In light of this tension between individual technical prowess and the social existence of source code, this article examines the communication processes involved in the construction, distribution and implementation of style guides for the JavaScript programming language within a contemporary software development environment—the collaborative platform GitHub[1].

While style guides and written documents have been enforced in formal, more traditional professional institutions heavily involved in writing (Christian et. al., 2011), GitHub presents a couple of specific aspects. First, as the most popular repository of open-source software, it is the locus of semi-formal participation and allows the copying and modifying of any project (*forking*). Second, large-scale private companies interact with distributed non-profit organizations and individual contributors in order to collaborate on software products, both free of charge and for-profit, which can then be modified by other users of the platform.

One of the most popular programming languages on GitHub is JavaScript[2]. However, the language itself lacks a clear, original style standard, and therefore has been the subject of various discussions on what a style guide should enforce and how it should enforce it. The active discussions taking place on the GitHub platform therefore represent a wide variety of opinions, skill levels, and institutional belongings. In this light, this article aims at investigating *which discursive strategies are used by programmers around the formation and implementation of style guides? How are those strategies affected by the specificity of a socio-technical environment such as GitHub? How are users of these style guides involved in these negotiations?* The answers to these questions will help qualify the nature of linguistic exchanges in a complex, digital-first, working environment and uncover the arguments and attitudes towards writing code.

To do so, I propose to look at three GitHub repositories, each representing different approaches to style guides: the JavaScript style guide as published by the Airbnb company, the popular, independent StandardJS and the exclusively format-oriented Prettier. A guiding criterion for narrowing our research field was to use the common indicators of stars (Borges & Valente, 2018) and *forks* (number of users having copied the repository to their own account, possibly to modify it further, and to submit these changes back into the original repository). For the JavaScript language, the top three most popular repositories are airbnb/javascript (100,000 stars, 19,500 forks),

[1] The sources for this article are all taken from comments from issues on three different GitHub repositories. When referencing these comments, the direct URL to the comment or to the general issue is given as a footnote, along with the status of the user (user, contributor or member).

[2] The State of the Octoverse, https://octoverse.github.com/, retrieved on 10.10.2020

prettier/prettier (37,400 stars, 2,500 forks) and standard/standard (24,000 stars, 1,900 forks) in December 2020. Additional JavaScript style guides include the IdiomaticJS style guide, the Google JavaScript Style Guide and the style presented by Douglas Crockford, but do not meet either the popularity criteria, or do not exist primarily on GitHub.

These repositories, while sharing the same effective outcome of providing a style reference for other JavaScript programmers, present differences in their approaches to achieving that goal, and in their organizational practices. The airbnb/javascript repository is the public-facing result of the guide used internally at the Airbnb company, a startup company privately valued at USD31 billion in 2019. It consists of a main document, README.md, the first page displayed on any GitHub repository, as well as configuration files for linters. The standard/standard repository is a non-profit open-source organization aiming at providing a non-modifiable, non-extensible way to programmatically enforce both stylistic choices and technical error-checking through the use of the *standard* software, a command-line utility which aims at automatically checking the style described on an additional RULES.md document. Finally the prettier/prettier repository limits itself to the strict formatting of source code and providing a command-line utility to automatically enforce and apply those changes.

This study focuses on the issues sections of each of these repositories. Traditionally used to keep track of bugs and technical enhancements for project maintainers, an issue can be created by and commented on by anyone with a GitHub user account but can only be closed (or « resolved ») by the original creator of the issue, or by project maintainers. While originally designed as a bug-tracker, research has shown that they are now the locus of more complex discussion, involving affective rhetoric and over-arching design inquiries, beyond specific technical fixes (Bissyandé et. al., 2013 ; Tsay et. al., 2014).

2. Methodology

The methodology employed here to analyze (1) how different actors negotiate the adoption of both subjective and objective stylistic norms, and (2) the influence of the technical environment in which those discussions take place, is based on the discourse analysis of individual issues in each examined repository.

This approach will focus mainly on the most commented issues, highlighting the discursive strategies and patterns of the participants; since the least-commented issues are related to specific issues and bugs which stand outside of the field of argumentation. Specifically, we aim at linking the meanings of the corpus not just to existing social realities (Bourdieu, 1982), but to kinds of communicative competences embedded in technical realities. Habermas provides a typology of arguments deployed through communicative action (Habermas, 1984). Among these are theoretical discourse (based on logic), practical discourse (based on situational appropriateness), aesthetic criticism and explicative discourse; we will see how effective each of these are, and particularly how practical discourse is influenced by the socio-technical environment in which the communication takes place. Additionally, we use here Habermas's notion of a negotiation as a bargain: in which both parties give up something in order to reach satisfaction, and contrast it with discussion, of which the result can be the change in stance of one, or both, parties. Regarding the restriction of the analysis to the exclusive locus of issues, we will consider that a read-only document (such as README.md) is relevant to our analysis insofar as it addresses a specific user, with a specific voice: if this linguistic interaction doesn't imply discussion, it certainly implies discourse.

Several studies have previously looked into the discourses on open-source communities (Berry, 2006), mostly through the use of large-scale computational techniques. The proposed approach examines specifically the nuances and implicit assumptions within the arguments deployed by the user, working under the hypothesis that, while broad strategies have been identified for issue resolution (Kavaler et. al., 2017), a more detailed and micro-level approach in the specific field of style guide adoption might provide insights in the daily practices of programmers, and to what extent they implement, re-appropriate, or hijack ("*détournent*") the broad strategies of technical resolution assumed by GitHub (De Certeau, 2011).

The gathered corpus consists of 12780 issues and pull requests, totalling 53673 comments. From these, we discarded the least commented issues, under the assumption that meaningful and diverse discursive interaction takes place after 15 or more comments, which leaves a core corpus of 80 issues for airbnb/javascript, 82 issues for standard/standard and 444 for prettier/prettier[3]. While these were the main point of focus, this study included cursory browsing of lesser issues, particularly the chronologically earlier ones.

As a complementary to the studies mentioned above, our approach is qualitative: reading the most commented issues through the lens of Habermas's arguments typology and evaluating them on multiple

[3] This data was gathered via GitHub's API; the script to generate the data can be found here: https://gist.github.com/periode/b7240e5797933d2dbae2dea30716a841

dimensions—the reference they resort to (the spatio-temporal, and causal contexts), the subject's stance (speaking as an individual or on behalf of a group), the user's background (shown by their status as either *Member, Contributor* or *User*, as by the other repositories of the corpus they might contribute to), and the ways that communication is initiated or concluded[4]. Reaction emojis, used to express passive agreement or disagreement with a message, are considered qualifying markers, not creating new meanings in and of themselves.

A limitation of this study is that the surveyed population might just be a motivated subset of users, putting emphasis on the correct automation of the correct style of their code. While the findings here might not therefore represent the broader software development communities, they nonetheless embody heightened forms of socio-technical interactions, which might be more faintly manifested in other developers.

In the first part, I will address the origin of style in programming and its need in contemporary, commercial programming practices, inscribing it further into both studies of style as social phenomenon, drawing on Simmel and Granger, as well as the anthropology of the written word, with Goody and Fraenkel. Building on this dichotomy, I then highlight the *intermediary objects* (Jeantet, 1998) that are code linters. I will conclude on an analysis of the GitHub development platform as a whole, its affordance for read-only and read-write documents, as well as its hosting of invisible communities, in the development of style guides.

3. TBD

3.1. The need for style in programming

The problem of style might be that « the practical existence of humanity is absorbed in the struggle between individuality and generality » (Simmel, 1991). Simmel's investigation of the topic focuses on the dichotomy between works of fine art and mass-produced works of applied arts[5]. Simmel draws a distinction between unique objects displaying the subjectivity of its maker, and the industrially produced and replicated objects, which are only meant to serve practical ends.

[4] Here, Jakobson's *phatic* function of language proves particularly useful.

[5] Incidentally, this is explicitly referred to on one of the examined repository, https://github.com/airbnb/javascript/issues/102#issuecomment-28157738, retrieved on 28.09.2020

As these two kinds of work exist at the opposite extremes of a single continuum, we can insert a third approach: that of the crafted object. It exists in-between, as a repeated display of its maker's subjectivity, destined for active use rather than passive contemplation (Sennett, 2009). So while style can be seen as a general principle which either mixes with, replaces or displaces individuality, style in programming doesn't stand neatly at either extreme. The work of Gilles-Gaston Granger, and his focus on style as a structuring practice can help to better apprehend style as a relationship between individual taste and structural organization (Granger, 1988); we will see how discussions around style guides are often the result of tensions between individual preferences and the process of structuring (programming) texts.

Historically, the emergence of style in computer programming is concomitant with the development of the software industry, starting in the structural shift of the 1970s (Djikstra, 1972), while retaining the personal, emotional attachment stemming from its relationship to craftsmanship. A self-proclaimed highly complex undertaking (Knuth, 1997), the understanding of source code by someone who hasn't written it (or, being the writer, hasn't read it in a while) is particularly difficult.

Programming style guides are then textual documents with both social and technical components. On the social side, they are only useful if unconditionally adopted by all members working on a particular code-base, since « all code in any code-base should look like a single person typed it, no matter how many people contributed.»[6]; in the strict sense, guidelines are therefore reference documents which should provide an answer to the question of what is the preferred way of writing a particular statement (e.g. *var* vs. *let*, or *camelCase* vs. *snake_case*). Beyond aesthetic preferences aimed at optimizing the clarity of a given source code, style guides also include a technical component which aims at reducing programming errors by catching erroneous patterns in a given codebase (e.g. variable declaration before initialization, loose reference to the function-calling context). This technical component, because it can be judged by an objective standard (i.e. bugs in a program), is however seldom the reason for internal disagreements within teams.

Finally, while coding style hasn't been explicitly shown to influence metrics by which programmer productivity is usually assessed (Cox and Fisher, 2008), it has nonetheless been linked to an improvement in program comprehension

6 From the guiding document of a JavaScript style guide, https://github.com/rwaldron/idiomatic.js/, retrieved on 29.08.2020

(Oman and Cook, 1988), and is regarded as a marker of quality work in the software development community[7].

This phenomenon of explicitly written rules, dependent no longer on their writer, but rather on the organization to which the writer belongs, presents similar patterns as those highlighted in the formalization of knowledge as it happened during the transition of societies from oral to written communication (Goody, 1986). For instance, the written codification of hitherto implicit, idiosyncratic rulesets, has had the result of further preventing modification of said rules, and elevating them from a personal reach to a universal one. However, the technical context for those studies of the impact of literacy is one in which the written word, once put down, is not easily modified.

The digital word, stored on and communicated via computers, presents two important differences. First, it can be as easily modified as it can be retrieved. Since one of the basic operations of the computing machine is copy and retrieval, anything that can be said on a digital medium always holds within itself the possibility of duplication, modification and, therefore, variation (Manovich, 2001). Second, source code is executable. Beyond the linguistic act of writing a statement, the result of this writing act can be automatically executed and enforced, without the need for social performance. Written code exists within these “chains of acts of writing” (Fraenkel, 2006), and once its validity has been confirmed and merged into active code bases, its enforcement is significantly easier than that of guides and protocols written in human languages (Brousseau & Moatty, 2003). Fraenkel’s concept will be considered in the context of the digitization and the automation of such writing acts: whether by the process of closing an issue, labelling it under a certain category, or automatically executing the rules of a style guide, without further human intervention.

We will see that intermediary objects (Vinck & Jeantet, 1995) play a crucial role in the productive activity of writing source code: as objects that lie in between several elements (source code, compilation process, IDE), several actors (programmers in the same team) and successive stages of a work process (drafting code, reviewing it, and committing it to production) , this study will rely on their concept to highlight the very active role that linters play in these negotiations.

3.2. Automatic writing with code linters

A final relevant element constituting our research field is the code linter. Present in every examined repository, and beyond the JavaScript ecosystem,

[7] See the article referred to on Prettier’s documentation: https://www.smashingmagazine.com/2012/10/why-coding-style-matters/, retrieved on 03.02.2021.

a code linter is a software which, given a set of syntactic rules, modifies one or multiple source code files to match said rules. This programmatic element, which can be seamlessly integrated into more complex workflows, represents both a transfer of agency from the human to the non-human, as well as an improvement in the systematic enforcing of style rules.

Due to the overwhelming presence of automatic tooling in modern development (Hilton et. Al, 2016), programming style guides make configuration files for linters one of the most essential parts of their project, effectively connecting human-readable texts with their machine-executable counterparts. The most popular of those linters in the JavaScript ecosystem is *eslint*, with over 13 million weekly downloads[8]. As a tool in a programmer's workflow, these linters with their associated configuration files are those intermediary objects (Latour & Woolgar, 1986), non-human actants contributing essentially to labour and knowledge-creation. Previous research in the sociology of work focused on how these intermediary objects affect the work processes (both conception, discussion and realization), particularly in establishing a framework within which work can take place (Vinck, 2009); coupled with the automated normativity of code (Lessig, 1999; Galloway, 2006), these linters represent a significant part of the impact of a style guide.

3.3. The GitHub Development Platform

Along with these low barriers to reproducibility and enforceability of code, another difference between organizations centered primarily on code and software and those in which code only constitutes a technical background is the porosity of the distinction between public and private. That is, when organizations maintain, or depend on, open-source software, the delimitation of which individual contributes to the organization's product becomes more fluid and temporary than in more traditional organizations (Hendry, 2008). At its most radical form, open-source developement is entirely non-hierarchical and horizontal (Benkler, 2006), a structure in which anyone can comment on the product and suggest modifications, even though actual contribution remains subject to additional social and economic constraints (Dabbish et. al., 2012). This configuration directly affects the scope of a style guide. If anyone can potentially contribute to any code base within a given language, then the scope for any style guide is that of universal adoption. This relatively loose set

[8] https://www.npmjs.com/package/eslint, retrieved on 03.08.2020.

of mutually-beneficial work relations constitutes the background of our research field and is accentuated by the specificity of the GitHub platform.

As all platforms, GitHub connects multiple actors and provides the backdrop for economic, social and cultural practices (Gillespie, 2010). Economically, GitHub provides a way to store, retrieve and modify text files (often source code) for distributed teams of contributors, grouped into projects (repositories), themselves administered by either personal or organizational accounts. Any public repository is accessible to anyone, and these projects can then be built upon and add to the value of a given commercial product. Socially, GitHub requires user registration to contribute to any of those repositories, and maintains a transparency policy which makes available all of a user's contributions on any public repository. These contributions overwhelmingly take the form of commits (direct modification of text files), pull requests (requests to an organization to integrate suggested changes to a text file), and issue creation and comment (asking or answering a question on a project repository). Culturally, user interactions on GitHub depend on agreed-upon practices and discourses, specifically when a user raises an issue, responds to it, or concludes (closes) it (Tsay et. al., 2014).

GitHub is also a combination of *read-write* texts and *read-only* texts. The read-only texts consist of the main text files of the repository, while the read-write texts are composed of all the discussions and suggestions taking place in the issues and pull requests sections. Drawing from literary theory, this distinction allows us to identify the more authoritative text and the more negotiable ones (Barthes, 1970).

We now turn to the analyses of the three repositories, in order to highlight which kinds of strategies are deployed by the parties taking part in discussions over style guide design and implementation.

4. TBD

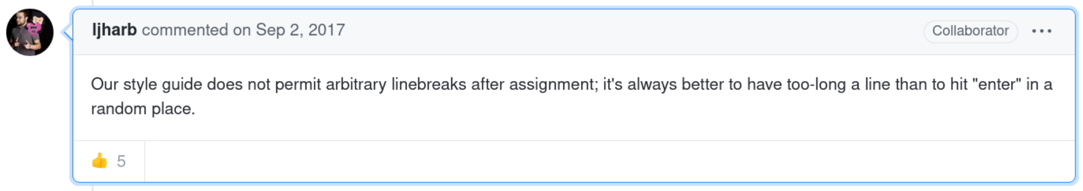
4.1. Airbnb

As pointed out by a user, the airbnb/javascript issues presents itself as much as a question-and-answer platform rather than as a traditional bug-reporting and fixing platform[9]: « Hi! First off, thanks for this package. Also, I'll preface this with: I'm not 100% sure I'm using the react/whitespace option correctly, so this is more of a question than a bug report. » (*mylestan*, user)

[9] https://github.com/airbnb/javascript/issues/1953#issue-379414953, retrieved on 03.08.2020.

The airbnb/javascript style guide consists of bug reports on eslint[10], individual projects[11] or the guide itself[12], even though issues that are labeled as "bugs" represent less than 1% of the total issues opened. In this case, bugs are understood as external inconsistencies between the *eslint* output and airbnb/javascript's guidelines. What would be deemed inconsistencies internal to the guide itself are labeled as *editorial*, reflecting a broader concern with communication and understanding of concepts, individual perspective, and paradigms over strict technical implementation issues.

Indeed, the unique aspect of airbnb/javascript is its dependency on the internal, well-organized structure of a private company. The constitution of the rules are, therefore, not up to debate. Such an implicit, private, out-of-reach existence of the origin of the guide is made explicit both by the creator of the publicly available guide, for whom the pronouns *we* and *our* do not refer to the open-source community of commenters and contributors, but rather to the internal team at Airbnb[13][14], as well as references by both parties in discussions to the practices of the company (« airbnb pushes for […] » (*KayakinKoder*, user)[15], « I'd like to see this covered by Airbnb's standards » (*rdsedmundo*, user)[16]) or to the codebase from which the style guide emerged [17].


ljharb commented on Sep 2, 2017

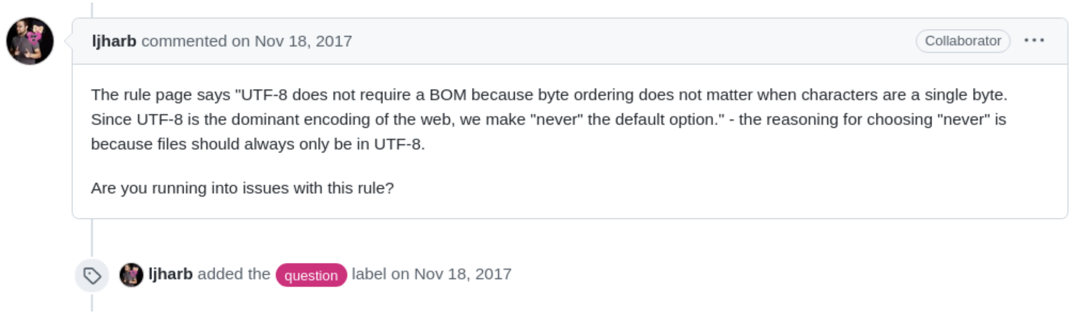
Our style guide does not permit arbitrary linebreaks after assignment; it's always better to have too-long a line than to hit "enter" in a random place.

On a chronological level, we see that actual discussions (mostly issues #1-#40) are those in which most of the interactions happen between Airbnb employees and take place as multi-sided discussions on how to improve the guide through group decision (« *I vote we go with* parseInt *because it's*

[10] https://github.com/airbnb/javascript/issues/1967, retrieved on 02.08.2020
[11] https://github.com/airbnb/javascript/issues/2261, retrieved on 02.08.2020
[12] https://github.com/airbnb/javascript/issues/828, retrieved on 02.08.2020
[13] https://github.com/airbnb/javascript/pull/455, retrieved on 02.08.2020
[14] https://github.com/airbnb/javascript/issues/1532#issuecomment-326755515, retrieved on 02.08.2020
[15] https://github.com/airbnb/javascript/issues/1365#issuecomment-350163209, retrieved on 03.08.2020
[16] https://github.com/airbnb/javascript/issues/1660#issuecomment-419087604, retrieved on 05.08.2020
[17] https://github.com/airbnb/javascript/issues/1185#issuecomment-262994841, retrieved on 04.08.2020

explicit. », hshoff, member)[18][19][20] or on bug- and tool-fixing[21]. What we see here is a private company using the tools and infrastructure of open-source processes embedded in GitHub to improve their productivity by writing internally consistent code (Kalliamvakou et. al., 2015), while, in a second moment, asking community members for their inputs without promising to implement them[22].

From this early period of a public interaction between fellow members of the same private organization, to the current situation of a large-scale interaction of distributed individuals and organizations over one of the most popular JavaScript style guides, the discursive strategies of both the maintainers and of the commenters and contributors have shifted. This is due in part to the fact that the creation of the styleguide is not collaborative, in the open-source sense of the term. Since a negotiation leading to a possible change of mind of the maintainer isn't possible, the alternative chosen by the Airbnb team is the didactic explanation of non-negotiable rules, shifting the interactions from a negotiation to an explanation, in which the outcome of the exchange is ultimately the alignment of the user to the views of the maintainer.



ljharb commented on Nov 18, 2017

The rule page says "UTF-8 does not require a BOM because byte ordering does not matter when characters are a single byte. Since UTF-8 is the dominant encoding of the web, we make "never" the default option." - the reasoning for choosing "never" is because files should always only be in UTF-8.

Are you running into issues with this rule?

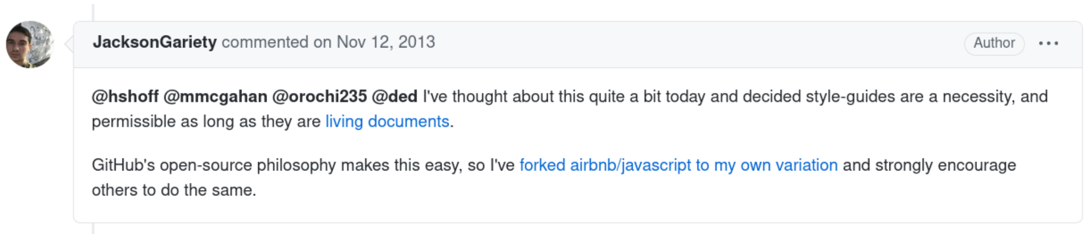
ljharb added the question label on Nov 18, 2017

One other example is featured in the explanation of an early issue opened on the style guide about the broader need for style guides; the creator responds with a pedagogical *métaphore filée* of painters and fine artists rather than logically, rationally approaching the need for consistent codebases[23].

Along with this desire for the maintainers of the project to explain, rather than discuss, the conclusion of this particular argumentation also sheds light on the forking mechanism as used in a discursive situation; forking acts as an end-all

[18] https://github.com/airbnb/javascript/issues/4, retrieved on 02.08.2020
[19] https://github.com/airbnb/javascript/issues/18, retrieved on 02.08.2020
[20] https://github.com/airbnb/javascript/issues/9, retrieved on 02.08.2020
[21] https://github.com/airbnb/javascript/pull/10, retrieved on 02.08.2020
[22] https://github.com/airbnb/javascript/issues/1089, retrieved on 05.08.2020
[23] https://github.com/airbnb/javascript/issues/102#issuecomment-28157738, retrieved on 05.08.2020

conclusion to the debate, in which another discussion is created, taking as its axiom the contentious proposals evoked in the base branch[24].


JacksonGariety commented on Nov 12, 2013 Author

@hshoff @mmcgahan @orochi235 @ded I've thought about this quite a bit today and decided style-guides are a necessity, and permissible as long as they are living documents.

GitHub's open-source philosophy makes this easy, so I've forked airbnb/javascript to my own variation and strongly encourage others to do the same.

Additionally, a particular aspect of airbnb/javascript is the progressive inclusion of *why* such a rule has been decided in the read-only README.md[25]. By coupling exhaustivity with justification, the main task remaining for the maintainers of the project is to continue the explanation of why things are the way they are[26].

The case of *prefer-default-exports* highlights the pattern of a maintainer repeating the same justification for recurring questions (« […] a default export is what a module *is*, and named exports is what a module *has*[...] »[27], « it's either what the module *is*, and thus the default export, or it's something the module *has*, and thus a named export »[28], « As to your specific question - a default export is what a module *is*, and a named export is what a module *has* »[29], *ljharb,* collaborator). Bugs reports are presented as misunderstandings from the commenter's point of view, asking the maintainers, not to justify their style choices, but rather to *explain* how one can write code that would better match the guidelines[30]. If that explanation fails, as in the *prefer-default-export* examples above, another strand of conversation is started by one of the core maintainers, user *ljharb*—himself a former Airbnb employee, and then member of TC39, the technical committee in charge of the design of ECMAScript, of which JavaScript is an offshoot. These conversations essentially consist in him providing *ad hoc* explanations when external users are confused about the purpose of a rule or, even further, in

[24] https://github.com/airbnb/javascript/issues/102#issuecomment-28259657, retrieved on 05.08.2020

[25] https://github.com/airbnb/javascript/issues/269, retrieved on 03.08.2020

[26] https://github.com/airbnb/javascript/issues/1880, retrieved on 07.08.2020

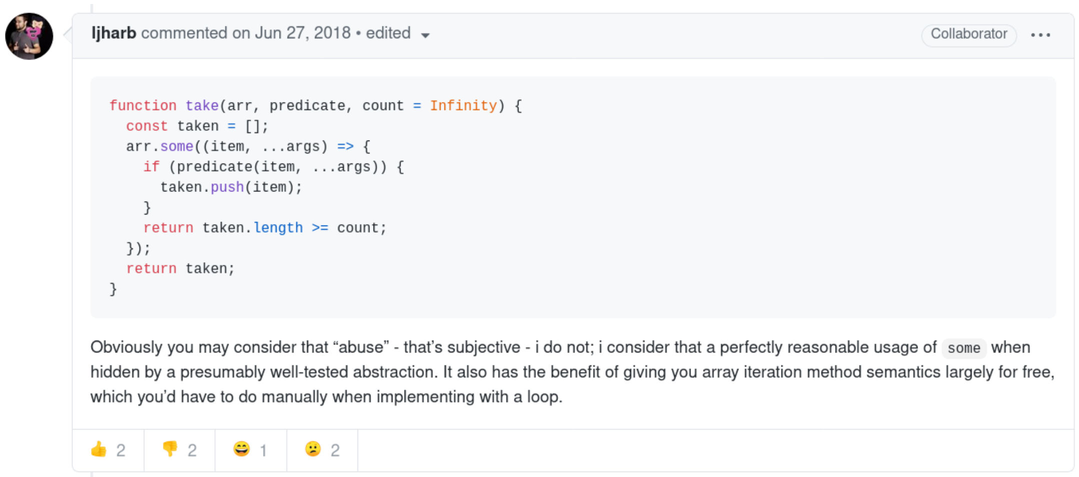
[27] https://github.com/airbnb/javascript/issues/2302#issuecomment-703436286, retrieved on 05.08.2020

[28] https://github.com/airbnb/javascript/issues/2191#issuecomment-596139441, retrieved on 08.08.2020

[29] https://github.com/airbnb/javascript/issues/1842#issuecomment-400194978, retrieved on 08.08.2020

[30] https://github.com/airbnb/javascript/issues/21, retrieved on 08.08.2020

re-organizing their code[31][32]. In this case, *ljharb* not only acts as a community manager rather than a project maintainer *per se*, but also eschews any discussions based on subjective preferences by providing a technical solution to any question asked, therefore showing that the airbnb/javascript style guide is not only exhaustive, consistent, but also implementable.



ljharb commented on Jun 27, 2018 • edited Collaborator

```
function take(arr, predicate, count = Infinity) {
  const taken = [];
  arr.some((item, ...args) => {
    if (predicate(item, ...args)) {
      taken.push(item);
    }
    return taken.length >= count;
  });
  return taken;
}
```

Obviously you may consider that "abuse" - that's subjective - i do not; i consider that a perfectly reasonable usage of `some` when hidden by a presumably well-tested abstraction. It also has the benefit of giving you array iteration method semantics largely for free, which you'd have to do manually when implementing with a loop.

👍 2 👎 2 😄 1 😕 2

This shift, from the stylistic to the technically feasible, is a recurring pattern which we'll also observe under different manifestations in the standard/standard and prettier/prettier projects.

In airbnb/javascript, then, the technical environment of GitHub issues have been repurposed from actual bug-tracking for Airbnb's employees, towards a didactic platform focused on explicative discourse, in a balance of both closed-source documents and open-source practices. The combination between one canonical document—uneditable beyond those with privileges, despite GitHub's pull request mechanism—which exhaustively covers almost all controversial use-cases in the language, along with a skilled maintainer explaining issues to the commenters individually, and, ultimately, the recourse to the process of forking as a concluding argument.

4.2. Standard

While airbnb/javascript provides a main README.md, with all the style rules immediately available at first glance, along with an *eslint* configuration file as the result of the closed-source work of a private company, standard/standard puts forth the *standard* software, an immediate, "out-of-the-box" solution which applies the project's rules to any file where the program is executed[33]—while

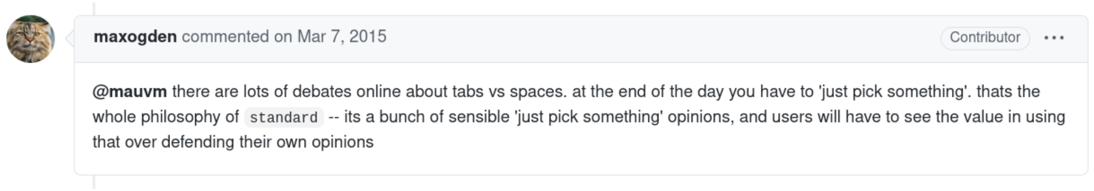
[31] https://github.com/airbnb/javascript/issues/1103#issuecomment-400711388, retrieved on 08.08.2020

[32] https://github.com/airbnb/javascript/issues/851#issuecomment-215479719, retrieved on 08.09.2020

[33] https://github.com/standard/standard/issues/94#issuecomment-87332587, retrieved on 13.08.2020

airbnb/airbnb only *describes*, standard/standard *implements*. The overall approach comes out of a more traditional open-source pattern, initiated and spearheaded by a single individual, *feross*, which then turned into a proper organization once enough community traction had been garnered[34][35].

The self-stated goal of standard/standard of « *no configuration* »[36][37] has led its maintainers (mainly *feross*, along with rotating members such as *linusU* and *rstacruz*) to inflect their discourse under the influence of technical efficiency, rather than rule justification as airbnb/javascript does. Indeed, the rules themselves are both stated by *feross* as well as discussed by the community—groups of users, members and contributors—for each release of the package. By leaving open the possibility of modifying their style rules, and subsequently making them immediately enforceable by their standard binary, the discursive focus isn't so much the number of discussions of so-called "religious" issues, known for the amount of devotion and heat that they attract (e.g. semicolons, « I was so caught up in negative thought loops that I did not think to research for community-forked modifications of "standard" that bless the use of semicolons. », *PythonProdigy*, contributor[38], along with parentheses [39][40], indentation[41]), but rather on technical implementation, alternative possibilities and community support, in a form of practical discourse. This emphasizes that the most productive issues are those based on a discussion around implementation, rather than a negotiation around style itself, again a prevalence of practical discourse over a theoretical, or aesthetic one.



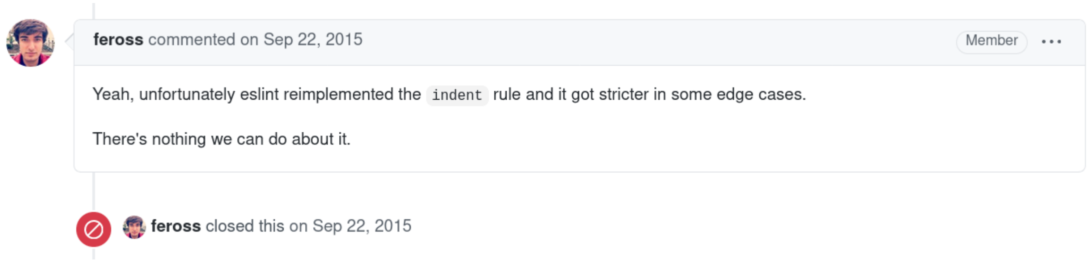
maxogden commented on Mar 7, 2015 Contributor

@mauvm there are lots of debates online about tabs vs spaces. at the end of the day you have to 'just pick something'. thats the whole philosophy of `standard` -- its a bunch of sensible 'just pick something' opinions, and users will have to see the value in using that over defending their own opinions

The first category of arguments laid out in the standard/standard repository is related to the actual, technical feasibility of the arbitrary[42], but not nonsensical,

[34] https://github.com/standard/standard/issues/846, retrieved on 13.08.2020
[35] https://github.com/standard/standard/issues/259#issuecomment-141881255, retrieved on 13.08.2020
[36] https://github.com/standard/standard/blob/master/README.md, retrieved on 10.08.2020
[37] https://github.com/standard/standard/issues/3#issuecomment-71952384, retrieved on 13.08.2020
[38] https://github.com/standard/standard/issues/962, retrieved on 13.08.2020
[39] https://github.com/standard/standard/issues/414#issuecomment-183459932, retrieved on 14.08.2020
[40] https://github.com/standard/standard/issues/219#issuecomment-170877700, retrieved on 14.08.2020
[41] https://github.com/standard/standard/issues/58#issuecomment-77710035, retrieved on 13.08.2020
[42] https://github.com/standard/standard/issues/3#issuecomment-71950165, retrieved on 15.08.2020

rules laid out in the README.md. The status of *feross* as « benevolent dictator » (*jprichardson*, member)[43] nullifies most of the arguments (e.g. « All very subjective though. I think @feross will just have to pick something :) » [44]), the remaining of which can sometimes be thoroughly assessed (« The reason being is that with only a single argument, the function is much easier to read without parens. Some folks who don't like this might argue about expansion and having a bigger diff when adding arguments down the road. », *KevinGrandon*, contributor)[45]. Given this status of most rules being somewhat indiscutable, upon what does the agreement of standard/standard rules rely? The status of the project as a convenient wrapper (i.e. a seamless *intermediary object* in a developer's workflow) around *eslint* poses that core technology as one of the final assessments of the validity of a given argument (e.g. « Probably better off making your case at eslint first, then coming back here » (*dcousens,* member) [46]). While this effectively avoids bikeshedding[47] in cases where the ultimate goal is to settle with one choice rather than debating multiple of them, this technological dependency also prevents some changes wished for by the community[48].


feross commented on Sep 22, 2015 · Member

Yeah, unfortunately eslint reimplemented the `indent` rule and it got stricter in some edge cases.

There's nothing we can do about it.

feross closed this on Sep 22, 2015

If the stylistic preference of a commenter still weighs more than the ease-of-use of the standard/standard package, then the strategy of the maintainers to solve this negotiation is to redirect them towards another part of the ecosystem, be it the direct configuration file which implements all of the

[43] https://github.com/standard/standard/issues/108#issuecomment-90990325, retrieved on 15.08.2020

[44] https://github.com/standard/standard/issues/309#issuecomment-152899586, retrieved on 15.08.2020

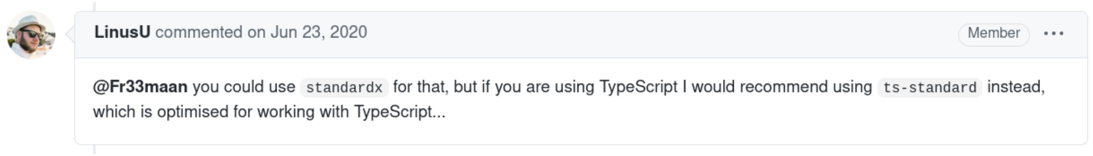
[45] https://github.com/standard/standard/issues/309#issuecomment-180208214, retrieved on 15.08.2020

[46] https://github.com/standard/standard/issues/720#issuecomment-266878246, retrieved on 15.08.2020

[47] https://github.com/standard/standard/issues/1356#issue-480058723, retrieved on 15.08.2020

[48] https://github.com/standard/standard/issues/257#issuecomment-142417059, retrieved on 15.08.2020

style rules[49], or the adoption of other packages maintained by the standard organization[50], bypassing the forking argument seen in airbnb/javascript.


LinusU commented on Jun 23, 2020 Member ···

@Fr33maan you could use `standardx` for that, but if you are using TypeScript I would recommend using `ts-standard` instead, which is optimised for working with TypeScript...

Different from the dismissive tone of the airbnb/javascript maintainer's repository, this redirection is an acknowledgement of the fact that opinions matter less than building a community, in line with the open-source ethos of group participation (e.g. the use of the *we* pronoun by *feross*[51]). This emphasis on community-building as a part of the overall strategy of standard/standard also affects their discursive strategy.

With a set of rules overwhelmingly decided upon by *feross*, some rational argumentations for the modification of these rules, the reliance on *eslint* as a ground for whether or not a rule can be applied, and the redirection of unsatisfied, or unconvinced community members towards other packages of the standard ecosystem (such as standard/standardx, standard/semistandard or standard/doublestandard) the last remaining kind of argument deployed for the adoption or rejection of style rules is the extent to which the community is using the existing rules.

On the one hand, this project has garnered visibility and is deeply embedded in an institutional network of projects using the standard/standard guide[52], including GitHub itself, and such a network provides credibility by association; these have been documented in order to further grow the community of users [53].

In practice, the biggest role that the community has in the negotiation of style guides is in whether or not any change in the guide would be a breaking change, and for how many projects using standard/standard, a requirement also known as backwards compatibility in the software industry. Used by project maintainers (« This is one of those decisions that we can't revisit. Nearly every repo that uses standard would break, and that's not acceptable –

[49] https://github.com/standard/standard/issues/240#issuecomment-224125128, retrieved on 15.08.2020

[50] https://github.com/standard/standard/issues/771#issuecomment-375609384, retrieved on 15.08.2020

[51] https://github.com/standard/standard/issues/1356#issue-480058723, retrieved on 15.08.2020

[52] https://github.com/standard/standard#who-uses-javascript-standard-style, retrieved on 11.08.2020

[53] https://github.com/standard/standard/issues/744#issue-200237553, retrieved on 12.08.2020.

even with major version bump. » (*feross*, member)[54], « Although personally I agree with the fundamental reasons that you argue for this. As it stands, this would be *so* much of a breaking change, it will never be accepted. » (*dcousens*, member)[55]). The closing argument is then the number of existing projects which would have to refactor their code in order to comply with the new rule, and is often the last comment to take place on an issue before that issue is closed[56][57].

The discursive strategies that the commenters and maintainers deploy in standard/standard revolve around issues of convenience, which have enabled a certain form of technical path-dependency[58]. The figure and work of *feross*, both agreeable and engaging in his exchanges and unilateral in his decisions, include in their discourse references to technical limitations in order to bypass subjective issues. Indeed, while the affordances of *eslint* are inherently digital, the adherence of a silent majority of standard/standard users is represented in a quantified manner, through unit tests resulting in a percentage of organizations and packages failing or passing said tests. Such an approach seems to manifest itself as the reification of community choices in order to increase its effective discursive power. Effectively, they rely on the extensive *adoption* of the rules, while airbnb/javascript relies on their extensive *applicability*.

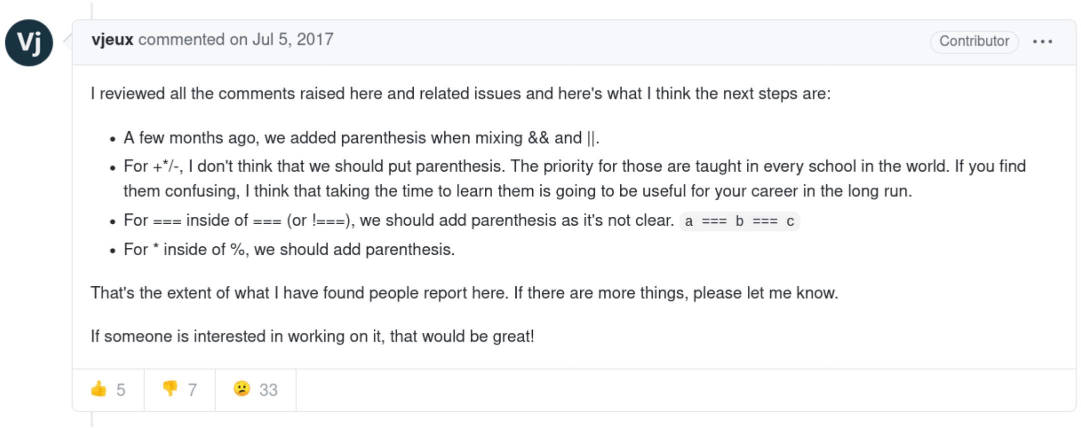
4.3. Prettier

Prettier is developed by Facebook employees, itself the maintainer of the *react* project, the most popular front-end development framework by the end of 2019[59]. Along with this institutional backing and internal success[60], prettier/prettier presents two specificities which differentiate from airbnb/javascript and standard/standard. First, it doesn't offer a clear and accessible style guide itself, and its README.md only contains one example to show the kind of work prettier/prettier does. Second, what it does is *essentially* different from the two previous examples, since it analyzes the

[54] https://github.com/standard/standard/issues/219#issuecomment-127446961, retrieved on 12.08.2020
[55] https://github.com/standard/standard/issues/240#issuecomment-135968036, retrieved on 12.08.2020
[56] https://github.com/standard/standard/issues/298#issuecomment-179571496, retrieved on 12.08.2020
[57] https://github.com/standard/standard/issues/720#issuecomment-515722463, retrieved on 13.08.2020
[58] https://github.com/prettier/prettier/issues/40#issuecomment-271769512, retrieved on 12.08.2020
[59] https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190, retrieved on 19.07.2020
[60] https://github.com/prettier/prettier/issues/5377#issuecomment-566173911, retrieved on 17.08.2020

source code, parses it and its inconsistencies and entirely reformats it according to internal rules—neither airbnb/javascript nor standard/standard offer default reformatting solutions. Such a language-independent approach (dealing directly with AST, short for *Abstract Syntax Trees*, a context-free structure) might explain its popularity across different languages, while the two other projects focus mainly on JavaScript. So, while not a style guide *de jure*[61] , it is one *de facto*, through its enforcement of rules, much more thorough than its alternatives.

Along with the absence of a clear, read-only, style guide announcing, if not explaining, the individual style decisions, the maintainers of the project (e.g. *jlongster*, *vjeux*, founders of the project while Facebook employees) tend to engage in conversations about the possibility to change existing styling rules[62] .


vjeux commented on Jul 5, 2017 Contributor

I reviewed all the comments raised here and related issues and here's what I think the next steps are:

- A few months ago, we added parenthesis when mixing && and ||.
- For +*/-, I don't think that we should put parenthesis. The priority for those are taught in every school in the world. If you find them confusing, I think that taking the time to learn them is going to be useful for your career in the long run.
- For === inside of === (or !===), we should add parenthesis as it's not clear. `a === b === c`
- For * inside of %, we should add parenthesis.

That's the extent of what I have found people report here. If there are more things, please let me know.

If someone is interested in working on it, that would be great!

👍 5 👎 7 😕 33

This willingness to change the behavior of their tool, along with its automation power and the founders institutional capital might explain the fact that the prettier/prettier repository has about four times more issues in total than the two other repositories examined, with more thoughtful, thorough and rational explanations[63], including detailed description of desired behaviour[64]. Still, these strategies do not always suffice to overcome personal preferences : « The solution is simpler than it looks: don't use Prettier » (odahcam, *collaborator)*[65]. In particular, prettier/prettier distinguishes itself by allowing contributors to open up discussions and negotiate changes not through pure discursive argumentation, but rather by offering implementations of an

[61] https://github.com/prettier/prettier/issues/5246#issuecomment-429788464, retrieved on 17.08.2020

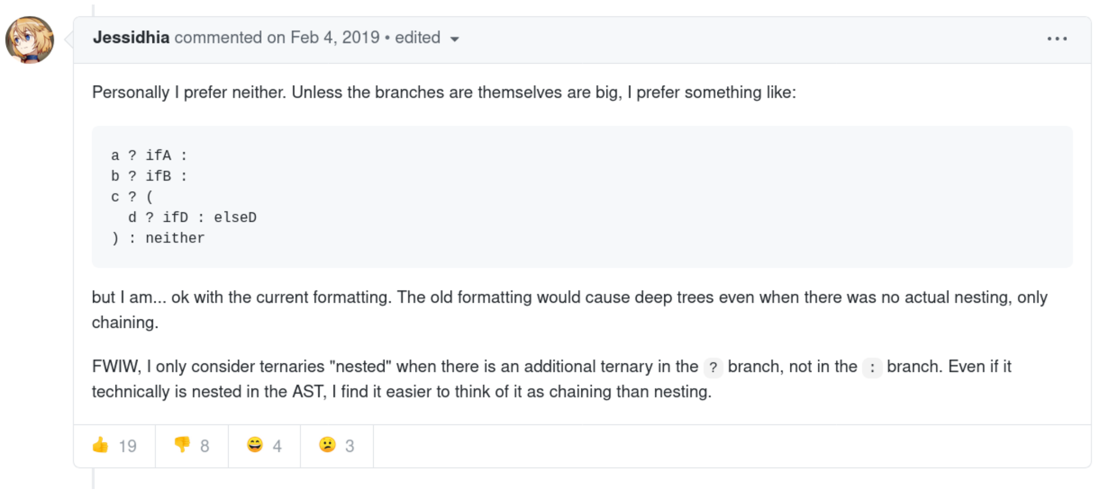
[62] https://github.com/prettier/prettier/issues/7884#issuecomment-605519802, retrieved on 18.08.2020

[63] https://github.com/prettier/prettier/issues/840#issuecomment-689978905, retrieved on 24.08.2020

[64] https://github.com/prettier/prettier/issues/3368#issuecomment-357312678, retrieved on 23.08.2020

[65] https://github.com/prettier/prettier/issues/4125#issuecomment-519068525, retrieved on 23.08.2020

alternative[66], including alternatives which concern "religious" issues[67]—in effect, long and elaborate argumentation always require *code* examples[68][69].


Jessidhia commented on Feb 4, 2019 • edited

Personally I prefer neither. Unless the branches are themselves are big, I prefer something like:

```
a ? ifA :
b ? ifB :
c ? (
  d ? ifD : elseD
) : neither
```

but I am... ok with the current formatting. The old formatting would cause deep trees even when there was no actual nesting, only chaining.

FWIW, I only consider ternaries "nested" when there is an additional ternary in the `?` branch, not in the `:` branch. Even if it technically is nested in the AST, I find it easier to think of it as chaining than nesting.

👍 19 👎 8 😄 4 😕 3

The process seems to be as follows: prettier/prettier formats a given code, the developer who might not agree with such a formatting result opens up an issue, argues for her opinion through the presentation of practical use-cases and actionable pull-requests and, depending on the technical soundness of the solution and the size of the community already using the package, might get their change accepted. Finally, while the thoroughness and politeness of most replies on this repository point towards a healthy community, it should be noted that the verbosity of some of the contributions are perceived as intimidating for comments who consider they English level sub-par (« I didn't read the whole conversation above because you guys really know how to speak English well slightly_smiling_face Makes me envy a bit » (*seahindeniz*, commenter)[70]). While prettier/prettier is the location of the most discussion of the three projects examined, the room left for discussion incidentally creates a space in which only those with communicative competence are heard.

Without a reference document, and with the recurring statement that "readability" and "preference" are subjective arguments, prettier/prettier also relies on technical arguments to solve debates, sometimes even before they

[66] https://github.com/prettier/prettier/issues/3368#issuecomment-374764905, retrieved on 22.08.2020
[67] https://github.com/prettier/prettier/issues/736, retrieved on 24.08.2020
[68] https://github.com/prettier/prettier/issues/7884#issuecomment-605898046, retrieved on 26.08.2020
[69] https://github.com/prettier/prettier/issues/5814#issuecomment-468978755, retrieved on 23.08.2020
[70] https://github.com/prettier/prettier/issues/7884#issuecomment-619147696, retrieved on 22.08.2020

happen. The seamlessness of the integration, as both a visible and invisible mediation, can eschew those debates altogether: « Hi @jlongster, just would like to chime in and let you know that your project has hit the jackpot and solved something that even (the self-proclaimed) Standard and Semi-Standard couldn't solve for me [...] Easy team-wide style enforcement that's based on an unbiased 3rd party algorithm to produce a one and only definitive way to indent absolutely each line of code »[71] (*gsklee*, user). This immediacy of programmatic action, and its adoption in return, is reminiscent of the power of interfaces in their ambiguity, in which a self-implementing tool acts as its own rationale (Galloway, 2012). Once again, the negotiation over the stylistic validity of the automated style guide is superseded by its immediate technical applicability.

Indeed, most of the official discourse of prettier/prettier (through its website and through oft-referenced conference talks) presents the project not as a style-guide *per se*, but as an AST-parser and printer. These definitions again steer the debate towards technical argumentation, rather than adoption by the community (such as standard), or explanation of practical rationale (such as airbnb); explicitly mentioning AST representation often becomes the grounds for the final stylistic decision (« it will be interesting to figure this out because the way it formats it now is because of how the AST is structured, which is based on operator precedence » (*jlongster*, member)[72][73][74][75]), and sometimes even used as counter-examples[76].

prettier/prettier's argument is that it is both opinionated and efficient, which has led to the early development of a "no-options" philosophy[77], a position which has moved from a read-write text, discussed by both maintainers and commenters, to a read-only text, as a canonical, if incomplete, text on the official website[78], used subsequently as an argument to respond negatively to requests for personal additions (« @JoshMcCullough We won't add options unless extremely necessary. Please read

[71] https://github.com/prettier/prettier/issues/40#issuecomment-271804674, retrieved on 21.08.2020

[72] https://github.com/prettier/prettier/issues/2#issuecomment-269872486, retrieved on 25.08.2020

[73] https://github.com/prettier/prettier/issues/5814#issuecomment-468827633, retrieved on 25.08.2020

[74] https://github.com/prettier/prettier/issues/9358#issuecomment-705940711, retrieved on 25.08.2020

[75] https://github.com/prettier/prettier/issues/187#issuecomment-281097771, retrieved on 25.08.2020

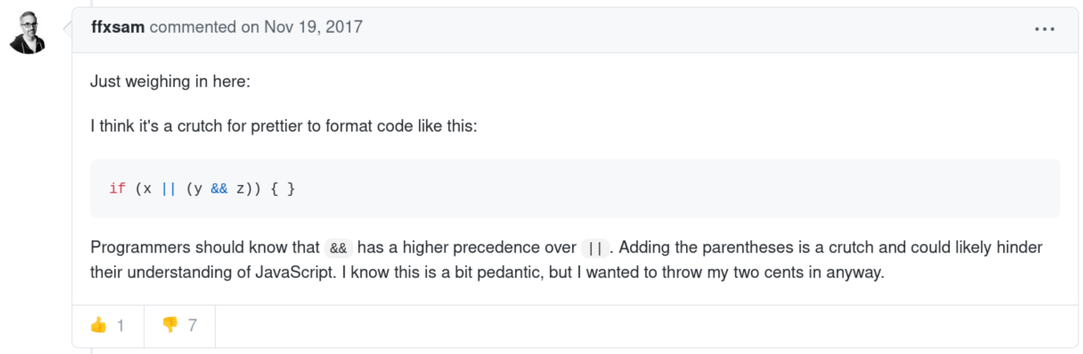
[76] https://github.com/prettier/prettier/issues/7884#issuecomment-605755932, retrieved on 26.08.2020

[77] https://github.com/prettier/prettier/issues/40, retrieved on 25.08.2020

[78] https://prettier.io/docs/en/option-philosophy.html, retrieved on 05.08.2020

https://prettier.io/docs/en/option-philosophy.html »[79][80][81] [diuailibe, *member*]), combined with a request of responding through programming languages, rather than human languages[82]. While standard/standard shares this approach of « no-options », their justify it by broadening the scope of what is considered acceptable (standard/standardx, standard/doublestandard), therefore growing the community, and not relying on the fact that only one single project should exist. Thus practices found in the airbnb project echo Goody's investigations of the immutability and canonization of written texts. Because the tool gets its value and usefulness from its invariability—and the communities which depend on its invariability[83]—, discussions around the tool itself can be made moot.

A final feature of the prettier/prettier discussions stems from both their exhaustiveness, the popularity of the project and the maintainers' decision to no longer accept options[84]. This results in several seemingly intractable discussions with over 100 comments, while the majority of other issues have less than 10 comments each[85]. These issues bring up the original tension of a style guide, in which each writer has an opinion[86] on what is the best way to solve a given dilemma[87], either as emotional, subjective statements[88] (« This issue is ridiculous. Just fix that » [*thalesfsp*, user][89]), or along with examples[90] and suggestions of pull requests.


ffxsam commented on Nov 19, 2017

Just weighing in here:

I think it's a crutch for prettier to format code like this:

```
if (x || (y && z)) { }
```

Programmers should know that `&&` has a higher precedence over `||`. Adding the parentheses is a crutch and could likely hinder their understanding of JavaScript. I know this is a bit pedantic, but I wanted to throw my two cents in anyway.

👍 1 👎 7

27.08.2020
[82] https://github.com/prettier/prettier/issues/840#issuecomment-482353176, retrieved on 27.08.2020
[83] https://github.com/prettier/prettier/issues/5377#issuecomment-646868677, retrieved on 27.08.2020
[84] https://github.com/prettier/prettier/pull/8540, retrieved on 25.08.2020
[85] https://github.com/prettier/prettier/issues/5377, retrieved on 23.08.2020
[86] https://github.com/prettier/prettier/issues/187#issuecomment-282690495, retrieved on 27.08.2020
[87] https://github.com/prettier/prettier/issues/187#issuecomment-345493031, retrieved on 25.08.2020
[88] https://github.com/prettier/prettier/issues/5377#issuecomment-669877250, retrieved on 25.08.2020.
[89] https://github.com/prettier/prettier/issues/187#issuecomment-385114296, retrieved on 25.08.2020
[90] https://github.com/prettier/prettier/issues/187#issuecomment-350849520, retrieved on 24.08.2020

The pattern we see here is that, due to the popularity of the project, first-time users weigh in on complicated debates, engage in the conversation with other commenters for a couple of replies, and then drop out of the discussion altogether. Coupled with the original intent of the maintainers to integrate community-argued changes, these issues remain open for all to chime in.

On the one hand, these discursive strategies seem to parallel the pattern of the dreaded bikeshedding[91], in which anyone can chime in, and including linguistic patterns such as « just my two cents », « IMHO » ("in my humble opinion"), « just gonna weigh in here », apparently. As such, they don't appear very productive due to repeated arguments[92][93] and they seem to affect the maintainer's ability to discuss (« The Prettier issue tracker naturally attracts and concentrates code style discussions. They've been moved from basically everywhere to here. I think this is a big factor to the maintainer burnout I've seen lately » [*lydell*, contributor] [94]).

On the other hand, however, the overall quality of arguments presented (combining use cases, opinions and responses to previous proposals) has led to a switch in the discursive strategy in which the discussion is summed up by the maintainers (« Summarizing a bit, it seems we have a handful of motivating examples and another handful of solutions now. », rattrayalex, *member)*[95][96][97] in order to provide moderation to the debate or to be pursued exclusively by the maintainers (« I've temporarily limited conversation the issue from spam/off topic so we can discuss between @prettier/core developers, I would like to hear everyone here, this problem has been present for a very long time and we need to give an official answer. » *alexander-akait*, member[98][99]). The result is that the prettier team, emerging from community-created discursive noise inherent to any popular open-source projects, engages less in terms of justifying the existing style, but in gauging

91 https://www.freebsd.org/doc/en/books/faq/misc.html#bikeshed-painting, retrieved on 15.07.2020

92 https://github.com/prettier/prettier/issues/187#issuecomment-383546996, retrieved on 26.08.2020

93 https://github.com/prettier/prettier/issues/5814#issuecomment-597403394, retrieved on 26.08.2020

94 https://github.com/prettier/prettier/issues/840#issuecomment-522295504, retrieved on 24.08.2020

95 https://github.com/prettier/prettier/issues/840#issuecomment-618186522, retrieved on 26.08.2020

96 https://github.com/prettier/prettier/issues/5814#issuecomment-469736563, retrieved on 26.08.2020

97 https://github.com/prettier/prettier/issues/5377#issuecomment-650607819, retrieved on 28.08.2020

98 https://github.com/prettier/prettier/issues/5377#issuecomment-669877250, retrieved on 28.08.2020

99 https://github.com/prettier/prettier/issues/5377#issuecomment-650153308, retrieved on 28.08.2020

how feasible an alternative is, once this discursive noise reaches a particular threshold, acting as a filter to turn quantitative input into a qualitative theoretical argument.

5. Conclusion

Style guides existing on GitHub have to tend to the specific issue of convincing individual users with strong subjective preference of adopting their recommendations in the midst of a technical environment which favors copying and customization. Starting from an approach of arguments based on the typology of communicative competence, we've highlighted that, beyond emotional and subjective statements as well as social justification of institutional origin or belonging, rational statements and commands are being affected by the technical *milieu* in which programmers write, read and work.

Particularly, we can see that there are multiple strategies which closely involve either source code examples, or executable code. Source code examples acts as a prime argument when human languages fail to communicate the objectivity of their statement, and is often required by all parties of a discussion to base their statements on. Still, those comments are often overlooked: the requirement of *readability*, crucial to a decision in style guides, re-appears in all its subjectivity when it comes to reading the source code example provided, pitting the argument in multi-sided subjective perspectives. In airbnb/javascript's case, when re-written code can act as an argument to the validity of the style guide, it is because previous code didn't comply, and became compliant again through *the maintainer*s work.

If source code seems to be a novel type of objective argument and yet remains trapped in subjective appreciation, executable code can act as its own argument. standard and prettier both rely on the fact that their style guides *work*, that they are efficient in what they do. Any change must first and foremost comply with the *feasibility* of the argument, no matter how sound the conceptual proposal is. As we've seen, prettier/prettier can replace a didactic guide by an efficient tool, while airbnb/javascript has to rely on a strict didactic approach, on top of providing an *eslint* configuration file, leading us to the consideration that code as actionable words is perhaps the most efficient in the adoption of a project, and therefore shifting a process of negotiation in which all parties are equal, to one in which the maintainer has a larger bargaining power and, more often than not, only accepts to discuss why such a decision has been made, and not whether it should be changed.

In terms of GitHub as a platform, we've observed a multiplicity of approaches, preventing a unilateral interpretation of how technical environments can entirely shape the nature of a discussion. GitHub's platform seems to be able to provide multiple kinds of discussions, ultimately geared by the social role of the maintainers and the ability to toggle between references to read-only (canonical) texts rather than keeping the discussion within read-write (discussion) texts.

Furthermore, GitHub does provide the interesting case of a highly-networked, highly-transparent working environment. This study has examined the role of a quantified community in argumenting for or against a particular stylistic choice, particularly present in standard/standard's strategy; it is necessary to mention the entangled nature of cross-references to other projects hosted on GitHub. Beyond naming existing alternatives[100], being able to reference another style guide on an issue[101], or to mention explicitly the name of a maintainer of one given style guide on a different one[102], facilitates the interactions of maintainers[103] and allows for an intricate web of intertextualities, allowing discussions happening on a given repository to act as arguments for another repository, or vice-versa. This added layer of complexity is beyond the scope of the current study, but offers a promising field for future research.

[100] https://github.com/prettier/prettier/pull/123#issuecomment-272029925, retrieved on 22.08.2020

[101] https://github.com/prettier/prettier/issues/3806#issuecomment-451615213, retrieved on 21.08.2020

[102] https://github.com/standard/standard/issues/811#issuecomment-294034120, retrieved on 04.08.2020

[103] https://github.com/prettier/prettier/issues/3503#issuecomment-352538566, retrieved on 21.08.2020

Bibliographie

BARTHES, Roland. (1970), *S\Z*, Éditions du Seuil, collection « Tel quel », p. 10.

BENKLER, Yochai (2006). *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. New Haven and London: Yale University Press.

BERRY, David M. (2004), « The Contestation of Code - A preliminary investigation into the discourse of the free/libre and open source movements » in *Critical Discourse Studies* Vol.1 pp. 65-89.

BISSYANDÉ, T. F., LO, D., JIANG, L., RÉVEILLÈRE, L., KLEIN, J., & TRAON, Y. L. (2013). « Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub », in *IEEE 24th International Symposium on Software Reliability Engineering* (ISSRE), pp. 188–197.

BORGES, H., & VALENTE, M. T. (2018). « What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform », in *Journal of Systems and Software*, vol. 146, pp. 112–129.

BOURDIEU, Pierre (2018). *Ce que parler veut dire*, Paris :Fayard, pp.3-5.

BROUSSEAU, E. et Moatty F. (2003). « Perspectives de recherche sur les TIC en sciences sociales: Les passerelles interdisciplinaires d'Avignon », in *Sciences de la société*, vol. 59, pp. 3-33.

CERTEAU, Michel de. (2011). « The Practice of Everyday Life ». Translated by Steven F. Rendall. 3rd ed. Berkerley: University of California Press.

CHRISTIAN, Darrell, JACOBSEN Sally A., & DAVID Minthorn. (2011)
« Associated press 2011 stylebook and briefing on media law ». New York: Associated Press.

COX, Anthony, FISHER, Maryanne. « Programming Style: Influences, Factors and Elements » in Second International Conferences on Advances in Computer-Human Interaction, pp.82-89.

DABBISH, L., STUART, C., TSAY, J., & HERBSLEB, J. (2012). « Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository » in P*roceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pp. 1277–1286.

DIJKSTRA, Edsger W. (1972) « Chapter I: Notes on Structured Programming. » In *Structured Programming*, Academic Press Ltd., pp. 1–82.

FRAENKEL, Béatrice. (2006). « Written Acts and Speech Acts: Performativity and Writing Practices », in *Études de communication*, no 29(1), pp. 69-93.

GALLOWAY, Alexander R., (2006). *Protocol: how control exists after decentralization*. Cambridge, Mass: MIT.

GALLOWAY, Alexander. R. (2012). *The Interface Effect*. New York :Polity.

GRANGER Gilles-Gaston (1988). *Essai d'une philosophie du style*. Paris, JACOB.

GILLESPIE, T. (2010). « The politics of 'platforms.' » in *New Media & Society*, vol. 12(3), pp. 347–364.

GOODY, Jack. (1986) *The Logic of Writing and the Organization of Society. Studies in Literacy, the Family, Culture and the State*. Cambridge, MA :Cambridge University Press.

HABERMAS, J. (1984). *The Theory of Communicative Action: Reason and the Rationalization of Society (Vol. 2)*. Boston, MA: Beacon Press.

HENDRY, Georges, (2008) « Public participation in proprietary software developmentthrough user roles and discourse », in *International Journal of Human-Computer Studies*, vol. 66,no. 7, pp. 545–557.

HILTON, M., TUNNELL, T., HUANG, K., MARINOV, D., & DIG, D. (2016). « Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects » in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 426–437.

JEANTET Alain. (1998) « Les objets intermédiaires dans la conception. Éléments pour une sociologie des processus de conception », in *Sociologie du travail*, 40e année n°3, Juillet- septembre 1998. pp. 291-316.

KAVALER, D., SIROVICA, S., HELLENDOORN, V., ARANOVICH, R., & FILKOV, V. (2017). « Perceived Language Complexity in GitHub Issue Discussions and Their Effect on Issue Resolution », in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pp. 72–83.

KERNIGHAN, BRIAN W., PLAUGER, Philip J. (1974). « Programming Style: Examples and Counterexamples. *ACM Computing Survey* Vol. 6, issue 4 (Dec. 1974), pp. 303-319.

KNUTH, Donald E. (1997) *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. (3rd Ed.) USA: Addison Wesley Longman Publishing Co., Inc.

LATOUR, Bruno, & WOOLGAR, Steve (1986). *Laboratory Life: The Construction of Scientific Facts*. Princeton, New Jersey: Princeton University Press.

LESSIG, Lawrence. (1999). *Code and Other Laws of Cyberspace*. New York City, NY:Basic Books, Inc.

MANOVICH, Lev. (2001). *The language of new media*. Cambridge, MA: MIT Press.

OMAN, Paul, COOK, Curtis (1988) *A Paradigm for Programming Style Research, ACM SIGPLAN Notices, vol. 23, pp.69-73.*

SENNETT, Richard (2009), *The Craftsman*, New Have, CO :Yale University Press, pp..45-47.

SIMMEL, Georg. (1991) « The Problem of Style. » in *Theory, Culture & Society* vol. 8, no. 3 (August 1991): pp. 63–71.

TSAY, J., DABBISH, L., & HERBSLEB, J. (2014). « Influence of Social and Technical Factors for Evaluating Contribution in GitHub » in *Proceedings of the 36th International Conference on Software Engineering*, pp. 356–366.

TSAY, J., DABBISH, L., & HERBSLEB, J. (2014). « Let's Talk about It: Evaluating Contributions through Discussion in GitHub » in P*roceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 144–154.

VAN DEN BOOGAARD, Adrienne (2008). « Stijlen van programmeren, 1952-1972 » in *Studium*, vol. 1 issue 2, pp.128–144.

VINCK, Dominique (2009). « De l'objet intermédiaire à l'objet-frontière: Vers la prise en compte du travail d'équipement » in *Revue d'anthropologie des connaissances*, vol. 3, issue 1, pp. 51-72.

VINCK Dominique & JEANTET Alain (1995). « Mediating and Commissioning Objects in the Sociotechnical Process of Product Design : a conceptual approach », in MacLean D., Saviotti P., and Vinck D. (eds.), Management and New Technology: Design, Networks and Strategy, Bruxelles: COST Social Science Series.pp. 111129.

Webographie

NATHER Edouard (1983). *The story of Mel, a Real Programmer*, FOLDOC, retrieved on 16.07.2020, URL : https://www.cs.utah.edu/~elb/folklore/mel.html

Appendix: Corpus of issues

airbnb/javascript

Issues are listed in their order of apparition in the manuscript.

ID	**Name**	**Date opened**	**Comments**	**Messages referenced**
102	Why does JavaScript need a style guide?	10 nov. 2013	43	1
...	...	...	...	...

standard/standard

Issues are listed in their order of apparition in the manuscript.

ID	**Name**	**Date opened**	**Comments**	**Messages referenced**
...	...	...	...	...
...	...	...	...	...

prettier/prettier

Issues are listed in their order of apparition in the manuscript.

ID	**Name**	**Date opened**	**Comments**	**Messages referenced**
...	...	...	...	...
...	...	...	...	...