How Ecosystem Cultures Differ: Perceptions of Values and Practices in 18 Software Ecosystems

Anonymous Author(s)

ABSTRACT

Software ecosystems have become one of the most important ways to organize software development, and to maintain and reuse code packages. But coordination can be a major challenge in software ecosystems when packages change, since packages tend to be highly interdependent yet independently maintained. The culture of an ecosystem includes those values and practices associated with managing change. We conducted a survey of thousands of developers in more than a dozen ecosystems, asking them about the values and practices that make up their communities' distinctive cultures. We identify what values and practices are shared among ecosystems (e.g. commitment to quality) and what differs between them (e.g. version numbering practices). We find that ecosystem values are not always what is advertised, and ties between values and practices are not easy to generalize. Our results lay the groundwork for future theory-building with the goal of helping ecosystem participants understand the space of possibilities for shaping and steering their communities.

1 INTRODUCTION

Software ecosystems—communities built around a shared programming language, platforms, and dependency management tools—have become one of the most important ways to organize software development, and to maintain and reuse code packages. Ecosystem style development is efficient, since common functionalities need only be developed, maintained, and tested by a single author or team, instead of many authors reimplementing the same functionality.

Coordination is a major challenge in software ecosystems, since packages tend to be highly interdependent yet independently maintained [3, 6, 12, 38]. In particular, sharing the same resources requires coordination when those resources change. Changes one developer makes to a shared package may affect many other people, for example creating new bugs or fixing old ones, introducing new features, or reorganizing or renaming components. Any of these actions may require rework from developers of software that uses that package.

A recent case study by Bogart et al. [6] investigated the role of ecosystem values, change practices, tools, and policies of three such ecosystems (Eclipse, R/CRAN, and Node.js/NPM). The study revealed stark differences among the three ecosystems: They had markedly different priorities (stability, rapid access to current research, and frictionless progress for developers, respectively), and their practices and policies reflected those priorities, for example in how they vet new changes and archive old revisions, how version numbers are specified and referred to, and what responsibilities the authors of packages have when making changes. These results are consistent with studies of open source and corporate *culture* [7, 13, 19, 24, 41, 44]: Culture arises from both values and practices which mutually reinforce each other, and culture has been shown to strongly influence performance [13]. To study culture in software ecosystems in depth (e.g., whether the formation and propagation of culture can be achieved without collocation and frequent informal contact assumed necessary in corporate settings [7]), we need to understand the current values and practices that developers hold within and across ecosystems.

We set out to take a broad look at ecosystem culture, collecting data on a range of values and practices. Specifically, we conducted a survey of thousands of developers in more than a dozen ecosystems, asking them about their perceptions of their ecosystem—its apparent values and how often things break—as well as their own values and development practices for their own packages. We were particularly interested in (1) to what degree ecosystems have a distinct culture of specific values and practices different from other ecosystems, (2) whether some values and practices are universal across ecosystems, (3) whether values and practices co-occur to reinforce each other, and (4) whether values with strong consensus are publicly visible and actively communicated.

We found that many ecosystems indeed have specific characteristics and differ significantly in a few values and practices from other ecosystems; the consensus on important values is often high. For example, *Stackage* strongly values *compatibility* among packages, which aligns with its origin and design goals as well as public statements about the ecosystem as we will discuss. At the same time, some values, especially *quality*, and some practices, including writing change logs and adding dependencies carefully, seem nearly universally independent of the ecosystem. Finally, while we found only a few a clear patterns among values and practices, we did find examples of publicly expressed values that are mirrored strongly by the community, as well design intentions that are not broadly communicated, and not reflected in the community.

Our insights provide a basis for studying culture of software ecosystems in depth. We show which ecosystems have unique characteristics, so that they can be used as a starting point for deeper explorations of the origins of values, both values that emerge organically and those that are manufactured through specific policy or tooling decisions. These explorations can be a basis eventually for learning how to nurture a strong culture for a sustainable community.

Overall we contribute a large-scale survey about ecosystem values and practices and an analysis of its answers across 18 ecosystems. Our results give a unique insights in the similarities and distinctive features of these ecosystems, and provide a starting point and research directions for further explorations on ecosystem culture. We will release the raw data of the survey responses (excluding text answers and demographic information for anonymity) together with the paper's publication for other researchers.

ESEC/FSE 2017, Paderborn, Germany
2016. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND AND RELATED WORK

Managing changes in software ecosystems. A software ecosystem is "a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them; ... frequently underpinned by a common technological platform or market?" [46]. A central piece of infrastructure found in most ecosystems is some means to package, version, and often host software artifacts, and to manage dependencies among them [1, 25, 28, 34, 43]. Change that propagates through a software ecosystem presents perhaps the greatest coordination challenge for this highly decentralized development model. Changes that are likely to break the functionality of packages that use them, often called "breaking changes", are common in practice [3, 5, 6, 8, 11, 16, 20, 22, 26, 29, 30, 36–38, 51].

Developers must therefore cope when exposed to breaking change from their dependencies; there is a large range of possible strategies, such as investing in regular rework, collaborating with upstream projects, not updating them (at the risk of missing bug fixes or security updates), or replicating functionality to avoid dependencies in the first place [6, 10, 42]. Package maintainers, in turn, have many ways to reduce the burden on their users; they can refrain from performing changes, clearly label of breaking changes, or help their users to migrate from old to new versions [6, 17, 35, 37].

Prior work has shown that practices, policies, and tools can shift the costs of breaking changes among ecosystem participants in different roles such as upstream developer, downstream developer, and end user [6]. These practices, policies, and tools differed dramatically among three studied ecosystems, each embedded in its own ecosystem culture, and reinforced by values held by participants.

Culture and values. This finding is consistent with studies of open-source communities, traditionally built up around individual projects, that expose open-source *culture* as a key ingredient of success [19, 41, 44]. In a corporate environment, strong corporate culture has long been shown to have a positive impact on performance [13], accounting for as much as 50 percent of the performance differential among competing firms [18]. Definitions of culture can vary somewhat, but they center around people sharing *values*, *practices*, and *vision* [7]. Values are traditionally seen as "important concerns and goals that are shared by most of the people in a group, that tend to shape group's behavior, and that often persist over time even with changes in group membership?" [24, p. 5]. When members adopt consistent sets of practices (or norms), reinforced by widely-held values, coordination and overall performance are improved.

Schwartz [40] characterized *values* as being beliefs that guide the selection of behaviors or evaluation of events, and as being inherently ordered by relative importance. Schwartz proposed a taxonomy and structure of personal life values and studied their relationship among many cultures. His empirical work involved surveys asking people to rate the importance of various values, using Likert scales running from "of supreme importance" to "not important" and finally "opposed to this value". Although he saw values as fundamentally ranked against each other, he collected value importance as ratings, since he [40] and others [2] point to difficulties for both survey taker and survey analyst in collecting rankings directly.

Whereas culture and values have been studied at societal scale in business [13, 18, 24, 39], communities [31], psychology [4], and ethics [9], they have been studied mostly in narrow contexts in software engineering: For example, Murphy-Hill et al. [32] found that game developers value creativity and communication with non-engineers more than application developers, manifested in different testing and architecture practices. Mair et al. [27] studied personal values of participants in the R community, though focused on personal motivations for participating, rather than value preferences for the ecosystem itself. citetizquierdo2015enabling have begun mapping the design space for change management practices. Overall though, little is known about how the policies and tools of a software ecosystem reflect or influence the values of the developers in the ecosystem's domain.

Culture may be actively influenced or fostered by policies and interventions. Practices that implement cross-project decision-making have sometimes been studied in open source and software ecosystems through the lens of *governance*, covering decisions from how to integrate third-party contributions [21], which model for decision-making is generally appropriate [23, 33], how open an ecosystem should be [49], and how people in different roles should be allowed to participate [50]. Tiwana et al. [45] describe abstractly how governance, architecture, and other factors cause ecosystems to evolve. O'Mahony and Ferraro [33] investigated the evolution of software ecosystem governance, but neither address how a community's values and policies allocate cost among participants. Such discussions focus mostly at a management level, without discussing code level decisions a developer might take. While some governance research in management contexts discuss the need for both evolvability and stability of an organization [48], they focus on general market mechanisms, but not on technical steps a policy might prescribe for evolving software packages. While Keertipati et al. study how technology decisions are made on Python [23], they focus on process conformance not on how decisions are reached.

Research questions. It is not clear how, and to what extent, ecosystem participants converge on cultural values and practices. In the corporate world, it is generally assumed that *collocation* and *frequent informal contact* are necessary for the formation and propagation of culture [7]. In order to understand the evolution and role of culture in software ecosystems, it seems useful to first get a broad picture of how practices and values are distributed within and across ecosystems. To the extent an ecosystem fails to develop a strong, coherent culture, or converges on some values and practices but diverges on others, the divergences may be potential problem areas, or opportunities for more effective coordination and collaboration. Hence, our first research question:

RQ1. Do ecosystem participants have a shared understanding of values and practices in their ecosystem?

It may be the case that the open source ecosystem style of development *requires* certain practices in order to be effective, and gives rise to certain values regardless of the details of the technology or identity of the ecosystem. To begin to understand which values and practices may be candidates for universal status, we investigate agreement and disagreement across different ecosystems:

RQ2. Do ecosystem participants agree on the importance of some values and practices across the spectrum of ecosystems?

It is plausible to expect that certain values and practices reinforce each other and are likely to occur together. For example, we would expect that ecosystems valuing stability adopt development practices that throttle change. We therefore investigate the relationship among values and practices across ecosystems:

RQ3. Do values and practices co-occur in mutually reinforcing patterns?

Finally, in some cases community values may be surprising to outside observers or newcomers to an ecosystem. We explore whether partitcipants recognize values implicitly from practices or because they are explicitly and publicly communicated, for example, through statements on the community's web pages or in talks:

RQ4. Are strong ecosystem values reflected in publicly visible communication?

3 METHODS

To address these questions empirically, we conducted a survey, asking questions of developers in a sample of ecosystems about perceived ecosystem values, personal values, ecosystem health, relative stakeholder power, motivations, and practices, focusing especially on perceptions and practices that would be more difficult to study by mining software artifacts in repositories. In this paper we only investigate the values and practices questions.

3.1 Survey design

The survey consisted of 108 questions; seven free text questions, three short blanks (ecosystem, package name and gender), and the rest multiple-choice scales. After an informed consent screen, participants first chose from a list of ecosystems, (or could write in another; we simply grouped these as "other" for analysis), then we presented blocks of questions in the following order: values, ecosystem health, stakeholder power, upstream practices, downstream practices, motives, and demographics.

Ecosystems. We considered ecosystems with a *network structure*, in which packages can depend on other packages and some infrastructure helps with sharing and compatibility. We started with a list of software repositories from Wikipedia[1] and added additional ecosystems with an active community that we could find.

We excluded ecosystems with a flat structure where packages depend only on a single shared platform (e.g., we excluded Android, where apps can interact through generic mechanisms rather than absolutely requiring another specific app) and ecosystems obviously too small to hope to get at least a few dozen responses. We also excluded ecosystems if they were different enough from the ones in the original Wikipedia list, that it was not possible to write clear questions that would apply across ecosystems. This excluded operating-system-level package managers like apt, rpm, and brew, and scientific workflow engines.

We conducted the survey with a list of 31 ecosystems. We recruited with a goal of at least 40 participants from each ecosystem. Afterward, we excluded 13 ecosystems from the analysis for which we did not receive at least 15 complete surveys: C++/Boost, Bower, Perl 6, Smalltalk, Tex/CTAN, Julia, Clojure/clojars, Meteor, Wordpress, SwiftPM, PHP's PEAR, Racket, and Dart/pub, leaving us with 18 ecosystems for our analysis: *Atom (plugins), CocoaPods, Eclipse (plugins), Erlang/Elixir/Hex, Go, Haskell (Cabal/Hackage), Haskell (Stack/Stackage), Lua/Luarocks, Maven, Node.js/NPM, NuGet, Perl/CPAN, PHP/Packagist, Python/PyPi, R/Bioconductor, R/CRAN, Ruby/Rubygems,* and *Rust/Cargo.*

Values. We took the values mentioned in the Bogart et al. study [6] as a starting point then systematically searched the web pages of all ecosystems for clues of other potential values. For example '*fun*' is mentioned as an explicit value in the Ruby community. [2] We assembled a list of 11 values with the following descriptions:

- *Stability:* Backward compatibility, allowing seamless updates ("do not break existing clients")
- *Innovation:* Innovation through fast and potentially disruptive changes
- *Replicability:* Long term archival of current and historic versions with guaranteed integrity, such that exact behavior of code can be replicated.
- *Compatibility:* Protecting users from struggling to find a compatible set of versions of different packages
- *Rapid Access:* Getting package changes through to end users quickly after their release ("no delays")
- *Quality:* Providing packages of very high quality (e.g. good security or correctness)
- *Commerce:* Helping professionals build commercial software
- *Community:* Collaboration and communication among developers
- *Openness* and Fairness: ensuring that everyone in the community has a say in decision-making and the community's direction
- *Curation:* Providing a set of consistent, compatible packages that cover users' needs
- *Fun* and personal growth: Providing a good experience for package developers and users

In the survey, we asked participants separately about the *perceived values* of the community—*"How important do you think the following values are to the <ecosystem> community?"* We used a seven point rating scale, adapted from Schwartz's value study [40]: *"extremely important", "very important", "important", "somewhat important", "not important", "community opposes this value",* and *"I don't know".* The first five options were separated visually from the last two to make clear that only the former were designed to approximate regular intervals (as recommended by Dillman et al. [15]).

In addition, we also asked participants a similar value question on the same scale about their *own values* with respect to a single package they worked on in the ecosystem. To encourage participants to think about concrete work that they are doing we asked for the name of a specific package that they worked on and used that package in the question: *"How important are each of these values in development of <package> to you personally?"*

Practices. In the *practices* part of the survey asked about many software engineering practices, many of which we mention throughout our analysis (Sec. 4 and 5); the full list and exact phrasing of

[1] https://en.wikipedia.org/wiki/Software_repository

[2] For example, in an interview Matsumoto said, *That was my primary goal in designing Ruby. I want to have fun in programming myself" [47].*

our questions can be found in Appendix A [3]. Surveyed practices encompassed the participant's personal practices and experiences with respect to documentation, support, timing, and version numbering for releases, selecting packages on which to depend, and monitoring dependencies for changes. These were asked either on an agreement Likert scale as above or on a frequency scale from *"never"* to *"several times a day"*. A subset of 15 questions relating to communication with developers of downstream packages were skipped for participants who indicated that they did not maintain a package used by others. To limit the length of the survey, we focused primarily on questions that cannot be answered or are difficult to answer by mining software repositories.

Demographics. We also asked *demographics* questions asking the participants' age, gender, role in the ecosystem, experience in open source and in software development generally, and computer-science education level.

3.2 Recruitment

We invested in significant outreach activities to recruit participants for the survey. First, we created a web page and twitter account to describe the state of current research in this area, in a form easily accessible to practitioners. We encouraged readers of the web page to take the survey to contribute additional knowledge about values in ecosystems. Second, we attended community events, including *npm.camp 2016*, to talk to developers and community leaders from multiple ecosystems about our research; several prominent community members tweeted about our web page and survey, resulting in surges of responses (CRAN and NPM particularly). Third, we promoted our web page and the survey in ecosystem-specific forums and mailing lists to *"developers who write <ecosystem> packages,"* hoping that our web page would spark interest in the topic. We also posted on twitter with hashtags appropriate for different ecosystems. Finally, for 21 ecosystems in which our outreach activity did not yield sufficient answers, we solicited individuals directly by email. After checking with Github to make sure our practices conformed with their terms of service, we sent 8,137 emails to package authors. We sampled these from packages in various ecosystems as culled from libraries.io, sending them a survey invitation if their package was shared on Github, they had chosen to make their emails public on that site, and they had not blocked their email address from ghtorrent.org's Github archive.[4].

Participants and their demographics. We succeeded in recruiting 2321 participants to take the survey between August and November of 2016. 932 of them completed the survey; however, we put value and health questions near the beginning, so there are more than 1200 answers to those questions. Statistical analysis of answers to early questions did not reveal any systematic differences between people who completed the survey and those who did not; as such we will report on answers to each section including any responses that got that far. We excluded as nonresponsive sections a person's response in which they rated all items exactly the same (and that rating was not an *"I don't know"* or *"Not applicable"*). We performed this test on eight sections of the survey, and the number of excluded blocks ranged from 11 (for a set of upstream practices) to 76 (for a set of downstream practices). When people were excluded from one block, their responses to other questions did not appear to be outliers, so we did not exclude entire people for this reason.

Respondees averaged 8.8 years of development experience, 7.2 years in open source, and 4.6 in the ecosystem they answered about. Slightly more than half (59%) had college degrees in CS. The most claimed role in the ecosystem was package lead developers (59%); Others ranged from the 8.5% who claimed a role in the founding or core team of the ecosystem, to 11% who only drew on ecosystem packages for their own projects. The average age was 33, with 152 18-24 year olds, and 6 over 65. Of those who gave their gender, 95.9% identified themselves as male, 3.2% as female, and 0.8% gave another gender.

3.3 Analysis

Display statistics. Figures 1, 2 and 3 were drawn by eliminating skipped or "don't know" values, merging "Not important" with "opposed to this value" answers, and drawing a violin plot, with a diamond symbol at the mean position. The violin bodies are smoothed, so the image portrays the mean and only a rough distribution.

Table 1 calculates highest ranked values for each ecosystem by identifying, for each person in the ecosystem, their highest rating of any of the 11 values, then incrementing a count for *all* values that person assigned that same rating to. The table lists the values with the highest three counts, and the consensus numbers are as described in the caption.

Qualitative analysis of text responses. 483 participants (21 %) gave an answer to at least one of seven free-response questions; 11 people gave answers to all seven. For the purposes of this study, we selectively analyzed text responses only to elucidate a few particular combinations of value and ecosystem that we discuss in the paper. To do so, we read all text responses in the relevant field for that ecosystem, and simply summarized or quoted the responses that were relevant. The quantity of responses were few enough within these subsets that more thorough coding and tagging methods seemed unnecessary.

4 OVERVIEW OF SURVEY RESULTS

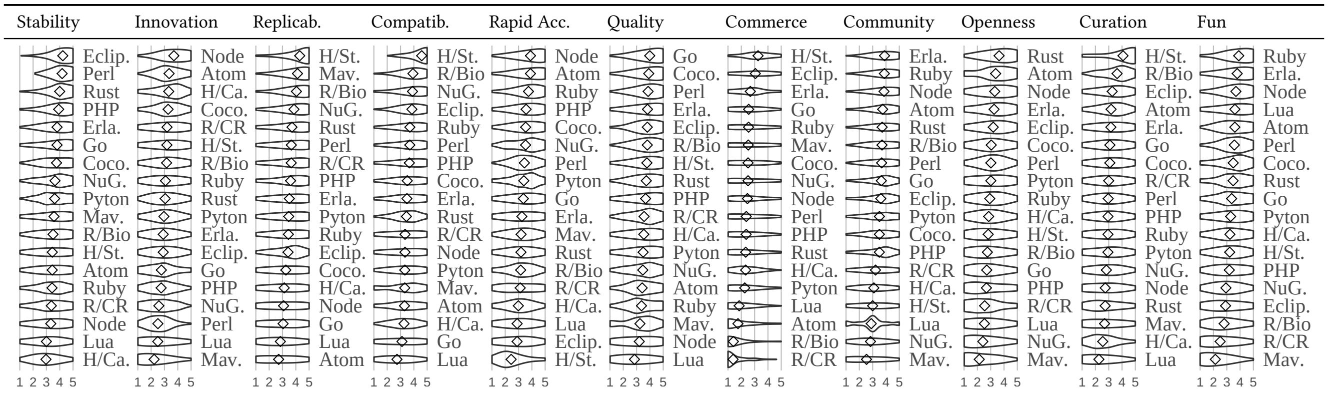
For additional context for exploring our research questions in the next two sections, we provide a brief descriptive overview of our survey results.

We plot answers to many of our survey questions in Figures 1, 2, and 3. Answers are often similar across many ecosystems and sorting them reveals often a spectrum without abrupt differences (e.g., it is rare that we see a clear division of ecosystems in two distinct groups). At the same time, the differences between ecosystems at either end of the spectrum are usually significant and often a few ecosystems stand out, as we will discuss.

Values (Fig. 1). Most values, except *commerce*, are considered important in most ecosystems. Stability, quality, and community are nearly universal values and *compatibility*, *rapid access*, and *replicability* are also rated highly across most ecosystems. Still, we see strong differences between ecosystems at each end of the spectrum.

[3] https://github.com/fse2017submission168/AppendixA/blob/master/fse2017-eco-survey.pdf

[4] We did not use githubtorrent.org's dataset except to extract this information



1: not important or opposed, 2: somewhat important, 3: important, 4: very important, 5: extremely important; plots show smoothed distribution; diamond indicates mean

Figure 1: Perceived community values; showing distribution of raw ratings, sorted by average to emphasize range of answers.

Personal values (shown in Appendix B [5]) correlate strongly with perceived community values, but developers, on average, rated *quality* as a much higher personally, compared to how they rated it as an ecosystem value; they also tended to rate *compatibilty*, *rapid access*, *curation* and *fun* slightly higher personally.

Change planning practices (Fig. 2). For change-planning practices, we see similar general patterns: Participants generally agree on the importance of many practices, but also the emphasis on certain practices varies in a spectrum with often significant differences between ecosystems at opposite ends of the spectrum. Specifically, participants across all ecosystems indicate that they perform breaking changes only rarely (typically less than once a year) and tend to document the changes and bundle multiple changes together to reduce interruptions. Also semantic versioning or similar versioning rules were adopted broadly across ecosystems, though different adoption rates between ecosystems were apparent. At the same time, the degree to which participants felt constrained to not make breaking changes differed by ecosystem, as did practices such as compromising on designs for backward compatibility and synchronizing with users before releasing changes. Some ecosystem-wide practices were, as expected, very distinctive for certain ecosystems—especially, coordinated releases in Eclipse and Bioconductor, gatekeeping in CRAN and Bioconductor, and versionless updates in Go.

Practices for coping with dependency changes (Fig. 3). When asked specifically about their package's exposure to breaking changes from upstream packages, participants across all ecosystems again reported low frequencies; only a quarter of our participants indicated more than one breaking change per year. Participants in ecosystems with more conservative change practices (e.g., *Eclipse, Erlang, Perl*) are exposed to slightly fewer breaking changes. Participants across all ecosystems indicated that they are conservative in adding dependencies and perform significant research first. In contrast, how they learn about updates (e.g., through personal contacts or tools), the rate to which they may skip them, and how they declare version constraints on dependencies depends significantly on the ecosystem.

5 ANALYSIS OF ECOSYSTEM CULTURE

5.1 RQ1: Shared understanding in ecosystems

We can infer whether an ecosystem has some notion of shared values or practices by looking whether these things differ from other ecosystems. If ecosystems had strongly shared culture that did *not* differ from other ecosystems, it would suggest the developers were really just part of a larger shared open source development culture.

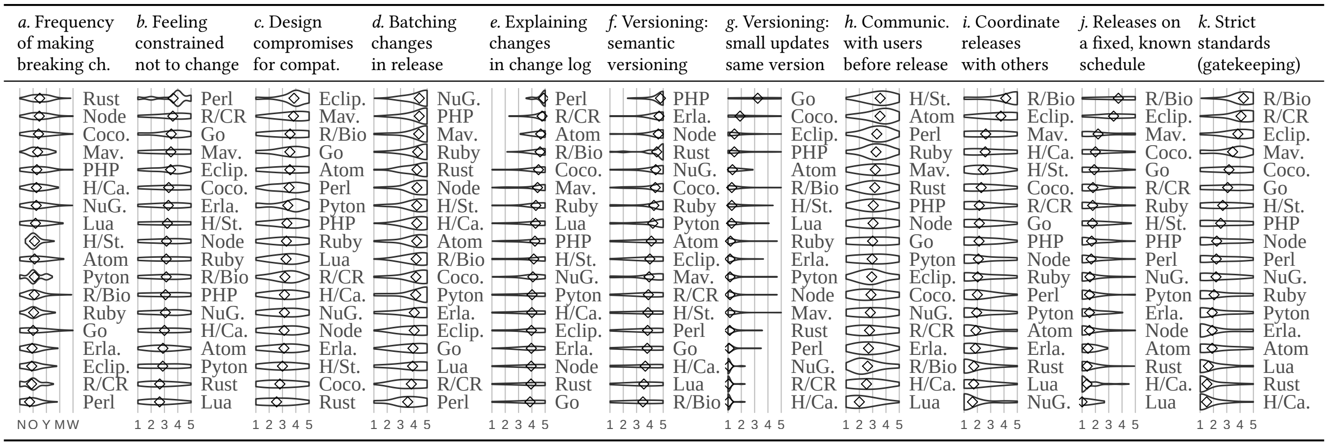
Our data suggests that ecosystems distinguish themselves from each other in a few key values and practices, rather than being unique across the board. There are many strong examples of this, including:

- *Bioconductor* and *Eclipse* stand out as coordinating releases on a synchronized and fixed schedule (Fig. 2i,j) and valuing *curation* (Fig. 1).
- *Go* has a distinctive version numbering practice that does not require version updates on all changes (Fig. 2g).
- *CRAN* and *Bioconductor* have strict requirements for submission and update of packages (Fig. 2k).
- *Lua* developers value *fun* and feel least constrained to make changes in their code and generally do not coordinate much with others (Fig. 2b,h,i).
- *Rust* has a strong stance on *openness* and makes least design compromises for backward compatibility (Fig. 2b,c).
- *Perl* developers universally commit on writing change logs (Fig. 2e).

Value differences by ecosystem are statistically significant for each of the values (ANOVA, p<.0001, F ranging from 3.348 for *quality* to 10.29 for *commerce*).

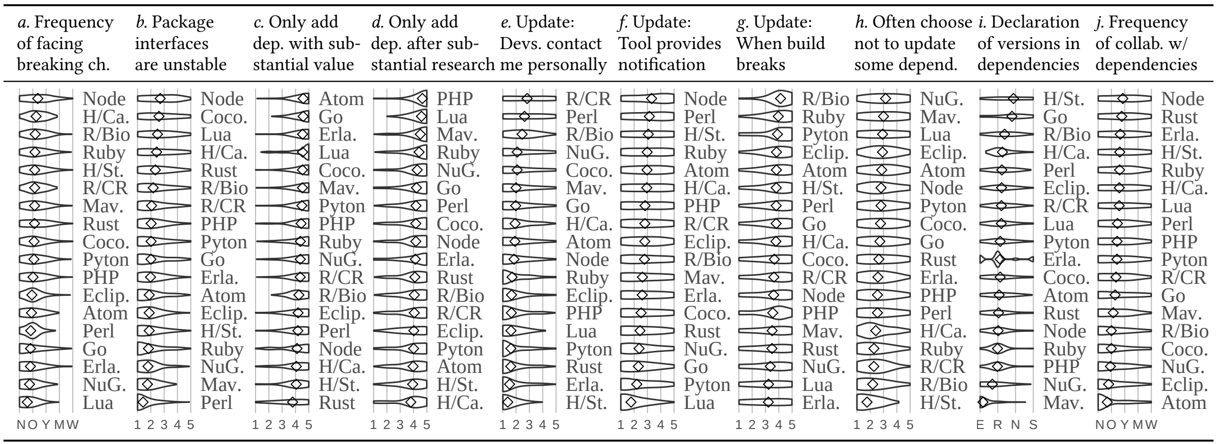
Ecosystems differed not just on what their developers valued, but also in the degree of consensus on those values. To quantify consensus, we considered each respondent's value ratings as a partial order ranking (with 11 values there were necessarily ties on our small scale), and counted how many ranked each value most highly. The results for the top three values in each community, shown in Table 1 point out how different ecosystems had different degrees of consensus.

[5] https://github.com/fse2017submission168/AppendixA/blob/master/fse2017-eco-survey.pdf



N: never, O: less than once a year, Y/M/W: several times a year/month/week; 1: strongly disagree; 3: neither agree nor disagree; 5: strongly agree

Figure 2: Practices of package maintainers and frequency of performing breaking changes.



N: never, O: less than once a year, Y/M/W: several times a year/month/week; 1: strongly disagree; 3: neither agree nor disagree; 5: strongly agree;
E: exact version number; R: version range; N: just by name; S: snapshot

Figure 3: Practices of package users and frequency of facing breaking changes.

There is evidence of broad consensus about the highest ranked value(s) for some ecosystems where a value clearly aligns with the core purpose of an ecosystem, for example:

- Stackage and Cabal/Hackage, the two Haskell-based ecosystems, contrasted strongly with each other in *compatibility* and *curation* with participants rating both of these values as much more important in Stackage, and less important in Cabal. Stackage was also rated markedly lower in *rapid access* than all other ecosystems. These values are consistent with the stated goals of Stackage ("to create stable builds of complete package sets"); Stackage was designed specifically to overcome problems with the previous Cabal/Hackage solution. Stackage is built on top of Cabal for the express purpose of curating compatible sets of versions—volunteers wait until a set of consistent package versions can be assembled, and release them as a unit, trading rapid release for tested compatibility. Stackage is not uncontroversial in the Haskell community and significant parts of the community still use the Cabal/Hackage ecosystem, likely emphasizing the perceived differences in values and practices among the two.
- *Maven* is primarily a build tool that comes with a centralized hosting platform for Java packages and was not designed as a collaborative platform. This purpose is reflected in strongly valuing *replicability* but least valuing *community*, *openness*, or *fun*.
- *Bioconductor* is a platform for scientific computation (specifically, analysis of genomic data in molecular biology) where *replicability* of research results is a key asset, but *commerce* is clearly not a focus.
- *Lua* is widely used as an embedded scripting language for games and prior work has shown that the culture of game developers is significantly different from that of application developers [32].

Table 1: Values most commonly rated highest, by ecosystem.

		Consensus in %		
Ecosystem	Top 3 values	C1	C2	C3
Haskell/Stack	compatibility > replicability > cur.	75	55	45
Perl/CPAN	stability > replicability > quality	64	40	31
Maven	replicability > stability > quality	64	38	32
Lua/Luarocks	fun > replicability > quality	64	35	17
Ecilpse	stability > compatibility > quality	62	48	37
NuGet	replicability > compat. > stability	59	37	20
Go	quality > stability > fun	56	37	19
R/Bioc.	replicability > quality > compat.	52	32	26
CocoaPods	quality > stability > compat.	52	30	17
Rust/Cargo	replicability > stability > comm.	51	31	23
PHP/Packag.	quality > stability > compat.	50	32	23
Node/NPM	rapid.access > community > innov.	50	24	15
Atom	rapid.access > fun > openness	50	26	17
Erlang	quality > fun > stability	46	24	18
Haskell/Cabal	quality > innovation > replicability	43	17	8
Python	replicability > quality > stability	42	20	14
Ruby	fun = community = rapid.access	41	18	12
R/CRAN	replicability > compat. > innov.	36	20	8

Consensus C*n* is the percent of respondents in each ecosystem who did not rate any value higher than any of the ecosystem's highest *n* values.

Others, like *R/CRAN*, have markedly less consensus, at least regarding the set of values that we surveyed.

Some, but far from all, practice differences can be explained by enforced policies or design choices in platform tools. For example *Maven* requires to specify exact versions on dependencies (Fig. 3i), *Bioconductor* and the core packages of *Eclipse* are released centrally (Fig. 2i,j), and *Bioconductor* and *CRAN* require reviews before packages are included in the repository (Fig. 2k). Some practices are supported by optional tooling in the ecosystem, such as tools to create notifications on dependency updates in the *Node.js* and *Ruby* community (Fig. 3i; e.g., *gemnasium* and *greenkeeper.io*). Other practices seem to be mere community conventions—for example, providing change logs is encouraged in the documentation of *CPAN* but not enforced, but the practices is universal (Fig. 2e). It is worth exploring the origin of such conventions and their relationship to values (see also RQ3 and RQ4).

Interestingly, there are some cases of practices with surprisingly little consensus in some ecosystems given what we know about tools and policies in that ecosystem. For example, a not-negligible set of *Node.js* respondents indicated that "package has to a meet strict standards to be accepted into the repository" (Fig. 2k), while that community's *NPM* repository does not have any checks and is in fact known for containing many junk packages. We conjecture, that such differences could come from two sources: First, ecosystem members may not be aware of the design space and what practices other ecosystems employ, such they have rather biased interpretations of what a "strict standard" is. Second, participants may be members in subcommunities values and practices that contrast from the larger community. For example there may be vetting of revisions among the developers within a specific project or subcommunity that is also hosted on *NPM*. A hint at this is that we found significant differences in some values (*opennness* and *curation*) depending the participant's role (project lead, founder, user, etc) and in more values (*innovation*, *community*, *opennness*, *curation*, *fun*) depending on their years of experience in an ecosystem (ANOVA, p<.05).

Summary and implications for future research. With regard to our research questions about shared understanding of values and practices within ecosystems, we can see that many ecosystems have clearly distinctions in a few key values and practices. Often the consensus on important values is high; some practices are actually enforced by policies and platform tools.

Our data provides insights in how ecosystems differ and which values and practices are characteristic. It is worth exploring reasons and origins for those differences beyond obvious differences in purpose and tooling. Our survey focused primarily on practices that cannot be inferred from software repositories, but it would be worth complementing our data with mined results, such as the frequency of testing or deprecating APIs (deprecation studies already exist for Smalltalk and Java [37, 38, 52] and could be generalized and compared across more ecosystems). Finally, studying subcommunities and how they relate in values and practices to the larger ecosystem can provide useful insights in how values and practices spread in an ecosystem.

5.2 RQ2: Agreement across ecosystems

Developers across all ecosystems valued *stability*, *quality*, and *community*, both in terms of personal values and and perceived community values. For *quality* in particular, developers felt even more strongly, and more consistently, that it was of high importance to them personally and to the ecosystem as a whole (the mean personal value of quality was about 0.8 scale points higher than the mean ecosystem value). The distribution of ratings *within* each ecosystem was particularly wide for the values *replicability*, *openness*, and *curation*, indicating generally less consensus on these values.

Breaking changes were infrequent, with a median of *less than once a year* for both the changes that our participants perform (Fig. 2a) and breaking changes that their package faces from dependencies (Fig. 3a). Also a number of practices are widely used across ecosystems: With a few notable exceptions, most developers use semantic versioning or comparable versioning strategies (Fig. 2f), batch multiple changes into a single release (Fig. 2d), document changes (Fig. 2e), and are conservative about adding dependencies to their projects (Fig. 3c). These seem to just generally be considered as good software engineering practices independent of programming language or ecosystem.

Summary and implications for future research. Regarding our research question, we can confirm that participants agree on the importance of a subset of the surveyed practices and values. Especially *quality* seems a near universal value for software engineers with little distinctions among ecosystems. Breaking changes are also generally avoided, though the strategies how this is achieved and as how difficult it is perceived depends on the specifics of the ecosystem.

Though quality seems a universal value, it would be interesting to what degree different ecosystems adopt different practices toward achieving quality (such practices were beyond the scope of our survey, but many can be publicly observed in repositories, policies, and discussions). Another open question is how common elements in the culture of open source ecosystems differ from cultures in other fields. For example, would a similar survey of corporate culture find such consistent personal support of quality, or are open source developers, as volunteers and engineers, more invested in these common values?

5.3 RQ3: Ties between values and practices

In order to investigate whether and how values and practices align, we investigated cases in which particular ecosystems differ markedly in values and practices from others; we thought that associations between practices and values where values are very strong would yield clear narratives that would generate hypotheses for checking more subtle relationships. In general, we did not find many clear associations of values and practices in our dataset. We focus our description on the top three ecosystems for two values with the clearest patterns.

Practices for achieving stability. The three ecosystems that valued *stability* most were *Eclipse, Perl,* and *Rust* (Fig. 1 and Tab. 1). Interestingly, participants in these ecosystems reported using very different practices to achieve stability. *Eclipse* relies on gatekeeping (Fig. 2k) and making design compromises to achieve backward compatibility (Fig. 2c); they police each others' backward compatibility and release together when they can be sure they will not break legacy code (Fig. 2i); developers feel constrained in making changes (Fig. 2b). *Rust*, in turn, ranked lowest in design compromises for backward compatibility (Fig. 2c), but high in semantic versioning (Fig. 2f); Rust's *Cargo* infrastructure insists on version ranges (not wildcards) for dependencies (Fig. 3i), letting users stay on an old version to attain stability, rather than putting the burden of backward compatibility on new development. Finally, *Perl*, unlike *Rust*, is low in semantic versioning (Fig. 2f), and in fact was the most likely ecosystem to claim they refer to dependencies by name only, not version number (Fig. 3i). They indicate some gatekeeping and design compromises but not to the extent of Eclipse (Fig. 2c,k). However in text comments, 12 of 30 participants who gave comments (40 %) mentioned Perl's extensive battery of tests run by volunteers. That is, there is no single strong pattern between *stability* and practices, but multiple different plausible ones.

Practices for achieving replicability. The three highest ranked ecosystems for *replicability* were *Maven*, *Bioconductor*, and *Stackage* (Fig. 1). As with stability, these ecosystems achieved their values differently. *Bioconductor* pursues a strategy like *Eclipse*: they release an update of a mutually consistent set of all ecosystem packages in regular half-yearly intervals (Fig. 2i,j). Because the whole ecosystem is synchronized, dependencies do not need a version number, and were more likely to say they referred to dependencies just by name (Fig. 3i). *Maven* approaches replicability through a more granular approach; it rated highest in referring to dependencies by exact version number (Fig. 3i; in fact all common Maven tooling enforces this), and it is common to continue to use old versions of dependencies (Fig. 3h). unlike *Bioconductor* where developers update the entire release to use current ones. Finally, *Stackage* takes a middle road by having third-party volunteers curate a set of compatible versions of packages; they used the "snapshot" strategy to dependency versioning the most (Fig. 3i), referring to packages by name but taking a snapshot copy to prevent impacts from changes.

Note that certain of these practices for *replicability* are common also in ecosystems that do *not* particularly value it. Notably, "snapshot" versioning was common in *Go*, even though *Go*'s infrastructure does not require or encourage the use of version numbering at all; but instead encourages upstream developers to write stable code (developers felt highly constrained , Fig. 2b). At the same time, it provides tools and encouragement for downstream developers to use "vendoring", which refers to copying a snapshot of the source code of dependencies into one's own repository. Replicability can still be achieved, but the burden is on the developers to assure this on their own. *Node.js* and *CRAN* provide similar snapshot tools, but they were used only occasionally in *Node.js* and only by a single *CRAN* developer among our participants. It seems taking snapshots is commonly a pattern to work around the fact that an ecosystem does not natively support and value *replicability*.

Other values. We expected to find other patterns, such as *community* and *openness* being opposed to gatekeeping practices, *innovation* and *rapid access* opposed to practices that slow down releases, such as scheduling or synchronizing releases and vetting contributions, or *fun* relating to less strict practices overall. While individual data points may support such hypotheses, we did not find consistent patterns.

Summary and implications for future research. With regard to our research question on co-occurrence of practices and values, though we expected some practices to plausibly related to the certain values, we do not see clear one-to-one relationships between values and practices. At least, there are choices among practices to pursue a value. Further research is needed to explain the full range of practice differences beyond these examples and to incorporate into that research practices that can be mined from software repositories. Research is also needed understand how ecosystems choose among the practice options, and to elaborate a more comprehensive model of the practices and their value tradeoffs, for example in the form of a pattern catalog for ecosystem design.

5.4 RQ4: Visibility of values

As part of a culture, practices and values are assumed to mutually influence each other [7, 24]. Some values may be explicit, either because the community has explicitly decided to pursue a value and to design practices correspondingly or because a community has acknowledged values that emerged from practices; other values might be implicit and shared by a large number of ecosystem members, though not necessarily visible to outsiders and newcomers. In this final part, we explore whether ecosystems with strong values reflect those in their public communication. For this exploration, we searched the official web pages of all ecosystems with strong values, looked for obvious statements in broadly visible publications about this ecosystem (e.g., Wikipedia), and looked for clues in textual responses to our survey.

Publicly stated values. Some ecosystems make public statements about values or expected practices. In several ecosystems this clearly aligns with perceived values and adopted practices.

- *Eclipse*'s developer documentation describes as a *"prime directive": "When evolving the Component API from release* *to release, do not break existing Clients"* [14]. Eclipse developers rated *stability* higher than any other ecosystem, and with the smallest variance (median = 'very important'; Fig. 1). Only 2 of the 56 Eclipse participants rated it lower than the midpoint, "important". The consensus among survey takers is strong (cf. Tab. 1) and consistent with the strong stance taken publicly by the Eclipse leadership.

- As discussed in Section 5.1, Haskell's *Stackage* ecosystem says on its front page *"Stackage is a stable source of Haskell packages. We guarantee that packages build consistently."* and on their Github page *"Stable Hackage: creating a vetted set of packages from Hackage"* which aligns well with the community's strong values of *compatibility* and *curation* (and highest consensus on these values; see Tab. 1). In addition, the community seems aware of its values and their tradeoffs (e.g., recognizing *rapid access* as a low priority for *Stackage*) due to public discussion around whether to adopt *Stackage* over the older *Cabal/Hackage*. Textual comments from participants in both ecosystems bore out the idea that the difference in value perception matched the *explicitly marketed contrast* between the two. Some preferred one over the other, while others saw them as complementary, and questioned our characterization of them as separate ecosystems. One respondent wrote: *"Stack/stackage completely changed the whole experience, though the community is still somewhat fragmented (some don't want to use it). Without using it, I spent a* lot *of time dealing with problems finding compatible dependencies. With it, it essentially never happens."*
- The *Ruby* community values *fun* as the highest value (Fig. 1). This value is not stated on Ruby's official web page, but can be found in many publications about Ruby, as mentioned in Sec. 3.1. Consensus (Tab. 1) is lower though than in many other ecosystems.

Inferred values. For several ecosystems with strong values, we could not find any obvious statement about those values. Instead, we conjecture that these values are implicit in the community and either derive from communication in community-internal channels or from common practices or technology decisions.

- *Perl/CPAN* values *stability* the most, and also as being perceived as the most stable. Unlike *Eclipse*, however, there are no imprecations on their web pages to avoid breaking interfaces. CPAN developers ranked highest of all ecosystems in feeling constrained from making breaking changes, and lowest in frequency of making breaking changes. We have only anecdotal evidence that might point toward a reason: One survey respondent attributed the stability to the inability to set a maximum version constraint on a dependency: *"The community builds backwards-compatible software because it must, not necessarily because it values backwards-compatibility above all things."* However another believed this value came from the personal values of Perl developers: *"the proportion of community participants with an operations/sysadmin background vs pure development is higher [... they] care strongly about making things stable and supportable."* Our survey participants for CPAN ranked as the oldest and most experienced as developers, with only 1 of the 43 respondents under 25, and 10 (21%) being under 35.
- *Maven* does not state any values explicitly on their web page and it is difficult to find value discussions in other public places. As discussed earlier (Sec. 5.1), *Maven* is described not as a community but as a build tool, which is reflected in values associated with build tools (*replicability*) and not with a community of developers (*community*, *fun*)
- The *LuaRocks* package manager's web page also does not state values explicitly. The web page of the *Lua* language itself mentions that *Lua* is the "leading scripting language in games", and promotes the language as "robust", "embeddable", "powerful", "small", and "fast'. Wikipedia also mentions extensibility and ease-of-use in development. Like *Ruby*, *fun* ranked as *Lua*'s most important value. We conjecture that this is related to its use in games, but the fun of *Lua* is not evidently promoted in its web presence.

The power of articulating values. The following two contrasting examples of *CRAN* and *Rust* illustrate that articulated values can influence a community, while unarticulated values can go unrecognized.

Unstated values go unrecognized: At least one core *CRAN* developer in Bogart et al.'s study [6], saw *CRAN*'s practices designed to support a value of "timely access to cutting edge research." We included the *rapid access* value specifically to explore whether the community agreed with it. This value seemed plausible for *CRAN* since it would explain why *CRAN*'s policy requires package authors to update their dependencies within a few weeks of a breaking change occurring, rather than allowing packages to rely on old dependencies.

However, survey participants did not perceive this value as particularly important for their ecosystem—*CRAN* in fact ranked in the bottom five in valuing *rapid access* (Fig. 1). It may be that without a knowledge of this design goal of *CRAN* and other considered design alternatives, *CRAN* package authors are more immediately aware of barriers to rapid access—namely the robust gatekeeping process (well recognized by participants, Fig. 2k), that channels new packages and package updates through human volunteers, who vet submissions thoroughly, and may reject them multiple times until criteria are met. If *rapid access* is indeed a value of the *CRAN* core team, it is not well communicated, and it is masked by the fact that the gatekeeping process, which serves other values, has the obvious effect of slowing down submission compared to popular other ecosystems without any reviewing.

Explicitly stated values trump inference: In *Rust*, the community seemed to agree with the founders' explicitly stated value of *stability* (Fig. 1), despite the fact that participants also perceived the ecosystem's packages to be unstable (Fig. 3b). The *Rust* language developers have been consistent in promising stability for the "stable" branch of the language,[6] to the extent that they test any compiler changes against the entire corpus of *Rust* programs they can find on Github. *Rust*'s *Cargo* archive also has stability-promoting technology, such as a "lock" file that records exact versions of dependencies, and a ban on using a wildcard allowing *any* version of a dependency (cf. Sec. 5.3 and Fig. 3j). But the analysis of their 2016 user

[6] e.g., https://blog.rust-lang.org/2014/10/30/Stability.html

survey[7] suggested why some users may see the ecosystem as unstable: Many packages ("crates") rely on unstable "nightly" development versions of the compiler to take advantage of interesting new features. They concluded that *"consensus formed around the need to move the ecosystem onto the stable language and away from requiring the nightly builds of the compiler."* This case illustrates how a community can perceive a strong community values when communicated properly, despite indications that practices may not yet align well.

Summary and implications for future research. With regard to our research question about the visibility of values, we found some ecosystems that made those values publicly explicit as well as ecosystems that held strong values without clear external clues. In one case community values were not as clear as the founders of the ecosystem may have hoped; in another a strong community value overcame the values people may have inferred from the ecosystem's practice outcomes. Our data alone cannot answer questions about the origin of values, but it opens many interesting research directions about how culture emerges in open-source ecosystems and how it can be nurtured.

Although our survey allows us only to explore the current values and practices of an ecosystems, we are interested in how distinct cultures of an ecosystem emerged. Did values emerge from intentional or accidental tooling decisions, were values intentionally promoted by ecosystem leads and merged into broader practice, or did they evolve over time, possibly from grassroots efforts in the community? Investigating such questions requires a longitudinal study, possibly analyzing past decisions and discussion and mining trends of practices from software repositories, as well as interviewing current and past stakeholders in an ecosystem. Our current survey can provide useful starting points for such a discussion and research on corporate culture can provide research methods for such exploration.

5.5 Validation of Bogart et al.[6]

We took the opportunity to validate some of the claims of a recent interview study on ecosystem values and practices [6], since that work interviewed only 28 people and we have available hundreds of survey results covering similar questions, and the same ecosystems, as that study.

Bogart et al. characterized practices and values of three ecosystems based on interviews with developers in each ecosystem. The values they inferred for *Eclipse* and *Node.js/NPM* align with our data: Eclipse participants did seem to value backward compatibility as postulated, *stability* and *compatibility* were their two highest ranked values (Tab. 1). Aligning with findings from the interviews, *Eclipse* developers were top-ranked in claiming to make design compromises in the name of backward compatibility (Fig. 2c). Aligning with the interview result that showed *Node.js* developers to value ease of contributions for developers, *Node.js* participants in our survey were top ranked in valuing *innovation* and ranked highly in both making frequent changes to their own package (Fig. 2a) and in facing breaking changes from dependencies (Fig. 3a), although they were mid-rank in feeling any less constrained from making changes than other ecosystems (Fig. 2b).

CRAN survey participants, as mentioned in Sec. 5.4, did not highly rank *rapid access* as expected from the interviews; and they were not more averse to adopting dependencies as predicted (not shown), although, as predicted, they did claim to clone code more (not shown). Aligning with interview results, they were top ranked in reporting being personally warned about changes in their dependencies (Fig. 3e), but, contrary to expectations, were low ranked in warning their own downstream users (Fig. 2h). As discussed above, it is possible that the interviews focused on the design intention of the *CRAN* team.

6 THREATS TO VALIDITY

As is typical of a survey, our sample could not be truly random; there may be selection bias relating to who we were able to reach via the venues we chose. We tried to mitigate this by sampling from forums, twitter, and direct mail. The survey was also quite long (and was advertised as such up front); people with less patience for long surveys, or less interest in questions of breaking changes, values, and practices, may have self-selected out. This could be significant if people with impatience for long surveys also have different software engineering practices and beliefs.

Another possible concern is that respondents may not have a broad perspective on what practices and values exist among ecosystems; they may rate practices as common or uncommon in their ecosystem, but this may be an ecosystem-specific notion about what standard they are comparing it against. For example an ecosystem that highly values stability may be more stable than others, but rate their ecosystem as unstable because their standards are so high. Future studies could calibrate such expectations against change data mined from repositories in the ecosystems.

We had difficulty recruiting sufficient participants from smaller ecosystems, like Perl 6 or Clojure; small ecosystems may have different characteristics than large ones. We do have some small ecosystems, Stackage and Lua, and they are outliers in some ways. So further exploration of small ecosystems, for example with interviews or analysis of artifacts, should be a priority for future work.

7 CONCLUSIONS

In this paper we investigated elements of software ecosystem culture that are adopted despite the lack of colocation and frequent meetings in which culture develops in business or other settings. We found that, (RQ1) there are indeed common values and practices that all the ecosystems we studied share; but (RQ2) distinctive cultures do develop although the depth and breadth of their consensus may vary. (RQ3) Values do not appear to completely determine practices; some values appear to motivate selection from a range of practices, but for other values the connections are obscure, at least for the values and practices we surveyed. Finally (RQ4) Some values are explicitly marketed and articulated; others seem to arise from elsewhere: perhaps a community's past choices, or perhaps from communication channels not obvious to outside observers.

We have also laid out suggestions for future research, including longitudinal studies of ecosystem decision-making, mining software repositories to explore practices and outcomes as measured, and theory building around the subtle relationships between values and practices. In the end, we believe a more rigorous understanding of the workings of software ecosystem communities will help these communities design policies and practices to further their values.

[7] https://blog.rust-lang.org/2016/06/30/State-of-Rust-Survey-2016.html

REFERENCES

[1] Pietro Abate, Roberto DiCosmo, Ralf Treinen, and Stefano Zacchiroli. 2011. MPM: A Modular Package Manager. In *Proc. Int'l Symp. Component Based Software Engineering (CBSE)*. ACM Press, New York, 179–188. DOI:http://dx.doi.org/10.1145/2000229.2000255

[2] Duane F Alwin and Jon A Krosnick. 1985. The measurement of values in surveys: A comparison of ratings and rankings. *Public Opinion Quarterly* 49, 4 (1985), 535–552.

[3] Cyrille Artho, Kuniyasu Suzaki, Roberto Di Cosmo, Ralf Treinen, and Stefano Zacchiroli. 2012. Why Do Software Packages Conflict? 141–150.

[4] Anat Bardi and Shalom H Schwartz. 2003. Values and behavior: Strength and structure of relations. *Personality and Social Psychology Bulletin* 29, 10 (2003), 1207–1220.

[5] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2015. How the Apache Community Upgrades Dependencies: An Evolutionary Study. *Empirical Software Engineering* 20, 5 (2015), 1275–1317.

[6] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2016. How to Break an API: Cost Negotiation and Community Values in Three Software Ecosystems. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*. ACM Press, New York.

[7] John Coleman. 2013. Six components of a great corporate culture. *Harvard Business Review* 5, 6 (2013), 2013.

[8] Bradley E Cossette and Robert J Walker. 2012. Seeking the Ground Truth: A Retroactive Study on the Evolution and Migration of Software Libraries. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*. ACM Press, New York, 55.

[9] Mary Crossan, Daina Mazutis, and Gerard Seijts. 2013. In search of virtue: The role of virtues, values and character strengths in ethical decision making. *Journal of Business Ethics* 113, 4 (2013), 567–581.

[10] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proc. Conf. Computer Supported Cooperative Work (CSCW)*. 1277–1286.

[11] Alexandre Decan, Tom Mens, and Maëlick Claes. 2017. An Empirical Comparison of Dependency Issues in OSS Packaging Ecosystems. In *Proc. Int'l Conf. Software Analysis, Evolution, and Reengineering (SANER)*.

[12] Alexandre Decan, Tom Mens, Maëlick Claes, and Philippe Grosjean. 2016. When GitHub meets CRAN: An Analysis of Inter-Repository Package Dependency Problems. *International Conference on Software Analysis, Evolution, and Reengineering* (2016), 493–504. DOI:http://dx.doi.org/10.1109/SANER.2016.12

[13] Daniel R Denison. 1990. *Corporate culture and organizational effectiveness.* John Wiley & Sons.

[14] Jim des Rivières. 2007. Evolving Java-based APIs. (2007). Online documentation: https://wiki.eclipse.org/Evolving_Java-based_APIs.

[15] Don A Dillman, Jolene D Smyth, and Leah Melani Christian. 2014. *Internet, phone, mail, and mixed-mode surveys: the tailored design method.* John Wiley & Sons.

[16] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J.S. Marron, and Audris Mockus. 2001. Does code decay? Assessing the evidence from change management data. *IEEE Trans. Softw. Eng. (TSE)* 27, 1 (Jan 2001), 1–12. DOI:http://dx.doi.org/10.1109/32.895984

[17] Johannes Henkel and Amer Diwan. 2005. CatchUp!: Capturing and Replaying Refactorings to Support API Evolution. In *Proc. Int'l Conf. Software Engineering (ICSE)*. ACM Press, New York, 274–283.

[18] James Heskett. 2011. *The culture cycle: How to shape the unseen force that transforms performance.* FT Press.

[19] Eric von Hippel and Georg von Krogh. 2003. Open source software and the "private-collective" innovation model: Issues for organization science. *Organization science* 14, 2 (2003), 209–223.

[20] Daqing Hou and Xiaojia Yao. 2011. Exploring the Intent Behind API Evolution: A Case Study. In *Proc. Working Conf. Reverse Engineering (WCRE)*. IEEE Computer Society, Los Alamitos, CA, 131–140.

[21] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2015. Enabling the Definition and Enforcement of Governance Rules in Open Source Systems, Vol. 2. IEEE, 505–514.

[22] Puneet Kapur, Brad Cossette, and Robert J. Walker. 2010. Refactoring References for Library Migration. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. ACM Press, New York, 726–738. DOI:http://dx.doi.org/10.1145/1869459.1869518

[23] Smitha Keertipati, Sherlock A Licorish, and Bastin Tony Roy Savarimuthu. 2016. Exploring decision-making processes in Python. In *Proc. Int'l Conference on Evaluation and Assessment in Software Engineering*. ACM, 43.

[24] John P Kotter. 1992. *Corporate culture and performance.* Simon and Schuster.

[25] Daniel Le Berre and Pascal Rapicault. 2009. Dependency Management for the Eclipse Ecosystem: Eclipse P2, Metadata and Resolution. In *Proc. Int'l Workshop on Open Component Ecosystems (IWOCE)*. 21–30. DOI:http://dx.doi.org/10.1145/1595800.1595805

[26] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2013. API Change and Fault Proneness: A Threat to the Success of Android Apps. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM Press, New York, 477–487.

[27] P. Mair, E. Hofmann, K. Gruber, R. Hatzinger, A. Zeileis, and K. Hornik. 2015. Motivation, values, and work design as drivers of participation in the R open source project for statistical computing. *Proceedings of the National Academy of Sciences* (2015). DOI:http://dx.doi.org/10.1073/pnas.1506047112

[28] Fabio Mancinelli, Jaap Boender, Roberto Di Cosmo, Jerome Vouillon, Berke Durak, Xavier Leroy, and Ralf Treinen. 2006. Managing the Complexity of Large Free and Open Source Package-Based Software Distributions. 199–208. DOI:http://dx.doi.org/10.1109/ASE.2006.49

[29] Michael Mattsson and Jan Bosch. 2000. Stability Assessment of Evolving Industrial Object-Oriented Frameworks. *Journal of Software Maintenance: Research and Practice* 12, 2 (2000), 79–102.

[30] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. 2013. An Empirical Study of API Stability and Adoption in the Android Ecosystem. In *Proc. Int'l Conf. Software Maintenance (ICSM)*. IEEE Computer Society, Los Alamitos, CA.

[31] Jonathan T. Morgan, Robert M. Mason, and Karine Nahon. 2011. Lifting the Veil: The Expression of Values in Online Communities. In *Proc. of the 2011 iConference.* ACM, New York, NY, USA, 8–15. DOI:http://dx.doi.org/10.1145/1940761.1940763

[32] Emerson Murphy-Hill, Thomas Zimmerman, and Nachiappan Nagappan. 2014. Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development? *Proc. Int'l. Conf. Software Engineering (ICSE)* (2014), 1–11. DOI:http://dx.doi.org/10.1145/2568225.2568226

[33] Siobhán O'Mahony and Fabrizio Ferraro. 2007. The emergence of governance in an open source community. *Academy of Management Journal* 50, 5 (2007), 1079–1106.

[34] Jeroen Ooms. 2013. Possible Directions for Improving Dependency Versioning in R. *The R Journal* 5, 1 (2013), 1–9. arXiv:1303.2140

[35] Tom Preston-Werner. 2013. Semantic Versioning 2.0.0. (2013). Online: http://semver.org.

[36] Steven Raemaekers, Arie van Deursen, and Joost Visser. 2012. Measuring Software Library Stability Through Historical Version Analysis. In *Proc. Int'l Conf. Software Maintenance (ICSM)*. IEEE Computer Society, Los Alamitos, CA, 378–387.

[37] Steven Raemaekers, Arie Van Deursen, and Joost Visser. 2014. Semantic Versioning versus Breaking Changes: A Study of the Maven Repository. In *Proc. Int'l Working Conf. Source Code Analysis and Manipulation (SCAM)*. IEEE Computer Society, Los Alamitos, CA, 215–224. DOI:http://dx.doi.org/10.1109/SCAM.2014.30

[38] Romain Robbes, Mircea Lungu, and David Röthlisberger. 2012. How do Developers React to API Deprecation? The Case of a Smalltalk Ecosystem. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*. ACM Press, New York, Article 56, 11 pages. DOI:http://dx.doi.org/10.1145/2393596.2393662

[39] Theresa Schmiedel, Jan vom Brocke, and Jan Recker. 2013. Which cultural values matter to business process management? Results from a global Delphi study. *Business Process Management Journal* 19, 2 (2013), 292–317.

[40] Shalom H Schwartz. 1992. Universals in the content and structure of values: Theoretical advances and empirical tests in 20 countries. *Advances in Experimental Social Psychology* 25 (1992), 1–65.

[41] Srinarayan Sharma, Vijayan Sugumaran, and Balaji Rajagopalan. 2002. A framework for creating hybrid-open source software communities. *Information Systems Journal* 12, 1 (2002), 7–25.

[42] Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. 2014. Software engineering at the speed of light: how developers stay current using twitter. *Proc. Int'l. Conf. Software Engineering (ICSE)* (2014), 211–221. DOI:http://dx.doi.org/10.1145/2568225.2568305

[43] Diomidis Spinellis. 2012. Package Management Systems. *IEEE software* 29, 2 (2012), 84–86.

[44] Katherine J. Stewart and Sanjay Gosain. 2006. The Impact of Ideology on Effectiveness in Open Source Software Development Teams. *MIS Q.* 30, 2 (June 2006), 291–314. http://dl.acm.org/citation.cfm?id=2017307.2017313

[45] Amrit Tiwana, Benn Konsynski, and Ashley a. Bush. 2010. Platform evolution: Coevolution of platform architecture, governance, and environmental dynamics. *Information Systems Research* 21, 4 (2010), 675–687. DOI:http://dx.doi.org/10.1287/isre.1100.0323

[46] Ivo van den Berk, Slinger Jansen, and Lútzen Luinenburg. 2010. Software ecosystems. *Proc. European Conference on Software Architecture Companion Volume (ECSA)* (2010), 127–134. DOI:http://dx.doi.org/10.1145/1842752.1842781

[47] Bill Venners. 2003. The Philosophy of Ruby: A Conversation with Yukihiro Matsumoto, Part I. http://www.artima.com/intv/rubyP.html. (2003).

[48] Jonathan Wareham, Paul B Fox, and Josep Lluís Cano Giner. 2014. Technology ecosystem governance. *Organization Science* 25, 4 (2014), 1195–1215.

[49] Joel West. 2003. How open is open enough?: Melding proprietary and open source platform strategies. *Research policy* 32, 7 (2003), 1259–1285.

[50] Joel West and Siobhán O'mahony. 2008. The role of participation architecture in growing sponsored open source communities. *Industry and innovation* 15, 2 (2008), 145–168.

[51] Wei Wu, Foutse Khomh, Bram Adams, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2016. An Exploratory Study of API Changes and Usages Based on Apache and Eclipse Ecosystems. *Empirical Software Engineering* 21, 6 (2016),

2366–2412.

[52] Jing Zhou and Robert J Walker. 2016. API Deprecation: A Retrospective Analysis and Detection Method for Code Examples on the Web. In *Proc. Int'l Symposium Foundations of Software Engineering (FSE)*. ACM Press, New York, 266–277.

A LIST OF QUESTIONS

For transparency and replicability, we list all evaluated questions of the survey including their exact phrasing. We exclude a small number of questions about power structures, community health, and motivation that we have not used in this paper.

Part I: Ecosystem.

- Please choose ONE software ecosystem* in which you publish a package**. If you don't publish any packages, then pick an ecosystem whose packages you use.
  * "Software ecosystem" = a community of people using and developing packages that can depend on each other, using some shared language or platform
  ** "Package": A distributable, separately maintained unit of software. Some ecosystems have other names for them, such as "libraries", "modules", "crates", "cocoapods", "rocks" or "goodies", but we'll use "package" for consistency.
  [selection or textfield, substituted for <ecosystem> in remainder of survey]

Ecosystem Role.

- Check the statement that best describes your role in this ecosystem.
  - I'm a founder or core contributor to <ecosystem> (i.e. its language, platform, or repository).
  - I'm a lead maintainer of a commonly-used package in <ecosystem>.
  - I'm a lead maintainer of at least one package in <ecosystem>.
  - I have commit access to at least one package in <ecosystem>.
  - I have submitted a patch or pull request to a package in <ecosystem>.
  - I have used packages from <ecosystem> for code or scripts I've written.
- About how many years have you been using <ecosystem> in any way?
  - < 1 year
  - 1 - 2 years
  - 2 - 5 years
  - 5 - 10 years
  - 10 - 20 years
  - > 20 years

Ecosystem values.

- "How important do you think the following values are to the <ecosystem> community? (Not to you personally; we'll ask that separately.)" — see Section 3.1 for the 11 value questions; **results shown in Figure 1**.
- How confident are you in your ratings of the values of <ecosystem> above?
  - Not confident
  - Slightly confident
  - Confident
  - Very confident
- "Is there some other value the <ecosystem> community emphasizes that was not asked above? If so, describe it here:"

Part II: Package.

- In the following we are going to ask about your experience working on one particular package. Please think of one package in <ecosystem> you have contributed to recently and are most familiar with. If you haven't contributed to a package in <ecosystem>, then name some software you've written that relies on packages in <ecosystem> packages. You may use a pseudonym for it if you are concerned about keeping your responses anonymous. — [text fields, substituted for <package> in remainder of survey]
- Do you submit the package you chose to a/the repository associated with <ecosystem>? (Choose "no" if the ecosystem does not have its own central repository.) — [yes/no]
- Is there any software maintained by other people that depends on the package you chose? — [yes/no]
- Is the package you chose installed by default as part of a standard basic set of packages or platform tools? — [yes/no]
- "How important are each of these values in development of <package> to you personally?" — see Section 3.1 for the 11 value questions; **results shown in Appendix B**.
- (OPTIONAL) Is there some other value important to you personally for <package> which was not mentioned? — [text fields]
- How often do you face breaking changes from any upstream dependencies (that require rework in <package>)? — **results shown in Figure 3a**
  - Never
  - Less than once a year
  - Several times a year
  - Several times a month
  - Several times a week
  - Several times a day
- How often do you make breaking changes to <package>? (i.e. changes that might require end-users or downstream packages to change their code) — [frequency scale as above] **results shown in Figure 2a**

Making changes to <package>.

- I feel constrained not to make too many changes to <package> because of
- potential impact on users. — **results shown in Figure 2b**
  - Strongly agree
  - Somewhat agree
  - Neither agree nor disagree
  - Somewhat disagree
  - Strongly disagree
  - I don't know
- I know what changes users of <package> want. — [agreement+don't know scale as above]
- If I have multiple breaking changes to make to <package>, I try to batch them up into a single release. — [agreement+don't know scale as above] **results shown in Figure 2d**
- I release <package> on a fixed schedule, which <package> users are aware of. — [agreement+don't know scale as above] **results shown in Figure 2j**

- Releases of <package> are coordinated or synchronized with releases of packages by other authors. — [agreement+don't know scale as above] **results shown in Figure 2i**
- When working on <package>, I make technical compromises to maintain backward compatibility for users. — [agreement+don't know scale as above] **results shown in Figure 2c**
- When working on <package>, I often spend extra time working on extra code aimed at backward compatibility. (e.g. maintaining deprecated or outdated methods) — [agreement+don't know scale as above]
- When working on <package>, I spend extra time backporting changes, i.e. making similar fixes to prior releases of the code, for backward compatibility. — [agreement+don't know scale as above]

Releasing Packages.

- A large part of the community releases updates/revisions to packages together at the same time. — [agreement+don't know scale as above]
- A package has to a meet strict standards to be accepted into the repository. — [agreement+don't know scale as above] **results shown in Figure 2k**
- Most packages in <ecosystem> will sometimes have small updates without changing the version number at all. — [agreement+don't know scale as above]
- Most packages in <ecosystem> with version greater than 1.0.0 increment the leftmost digit of the version number if the change might break downstream code. — [agreement+don't know scale as above]
- I sometimes release small updates of <package> to users without changing the version number at all. — [agreement scale, without 'don't know'] **results shown in Figure 2g**
- For my packages whose version is greater than 1.0.0, I always increment the leftmost digit if a change might break downstream code (semantic versioning). — [agreement as above] **results shown in Figure 2f**
- When making a change to <package>, I usually write up an explanation of what changed and why (a change log). — [agreement as above] **results shown in Figure 2e**
- When working on <package>, I usually communicate with users before performing a change, to get feedback or alert them to the upcoming change. — [agreement as above] **results shown in Figure 2h**
- When making a breaking change on <package>, I usually create a migration guide to explain how to upgrade. — [agreement as above]
- After making a breaking change to <package>, I usually assist one or more users individually to upgrade. (e.g. reaching out to affected users, submitting patches/pull requests, offering help) — [agreement as above]

Part IV: Dependencies.

- In the last 6 months I have participated in discussions, or made bug/feature requests, or worked on development of another package in <ecosystem> that one of my packages depends on. — [yes/no]
- Have you contributed code to an upstream dependency of one of your packages in the last 6 months (one where you're not the primary developer)? — [yes/no]
- About how often do you communicate with developers of packages you depend on (e.g. participating in mailing lists, conferences, twitter conversations, filing bug reports or feature requests, etc.)? — [frequency scale, as above] **results shown in Figure 3f**

For most dependencies that my packages rely on, the way I typically become aware of a change to the dependency that might break my package is:

- I read about it in the dependency project's internal media (e.g. dev mailing lists, not general public announcements) — [agreement scale, as above]
- I read about it in the dependency project's external media (e.g. a general announcement list, blog, twitter, etc) — [agreement scale, as above]
- A developer typically contacts me personally to bring the change to my attention — [agreement scale, as above] **results shown in Figure 3e**
- Typically I get a notification from a tool when a new version of the dependency is likely to break my package — [agreement scale, as above] **results shown in Figure 3f**
- Typically, I find out that a dependency changed because something breaks when I try to build my package. — [agreement scale, as above] **results shown in Figure 3g**
- How do you typically declare the version numbers of packages that <package> depends — **results shown in Figure 3i**
  - I specify an exact version number
  - I specify a range of version numbers, e.g. 3.x.x, or [2.1 through 2.4]
  - I specify just a package name and always get the newest version
  - I specify a range or just the name, but I take a snapshot of dependencies (e.g. shrinkwrap, packrat)
- What is the common practice in <ecosystem> for declaring version numbers of dependencies? — [same scale as previous + "don't know"]

Using or avoiding dependencies.

- When adding a dependency to <package>, I usually do significant research to assess the quality of the package or its maintainers, before relying on a package that seems to provide the functionality I need. — [agreement scale, as above] **results shown in Figure 3d**
- It's only worth adding a dependency if it adds a substantial amount of value. — [agreement scale, as above] **results shown in Figure 3c**
- I often choose NOT to update <package> to use the latest version of its dependencies. — [agreement scale, as above] **results shown in Figure 3h**
- When adding a dependency, I usually create an abstraction layer (i.e., facade, wrapper, shim) to protect internals of my code from changes. — [agreement scale, as above]
- When working on <package>, I often copy or rewrite segments of code from other packages into my package, to

avoid creating a new dependency. — [agreement scale, as above]

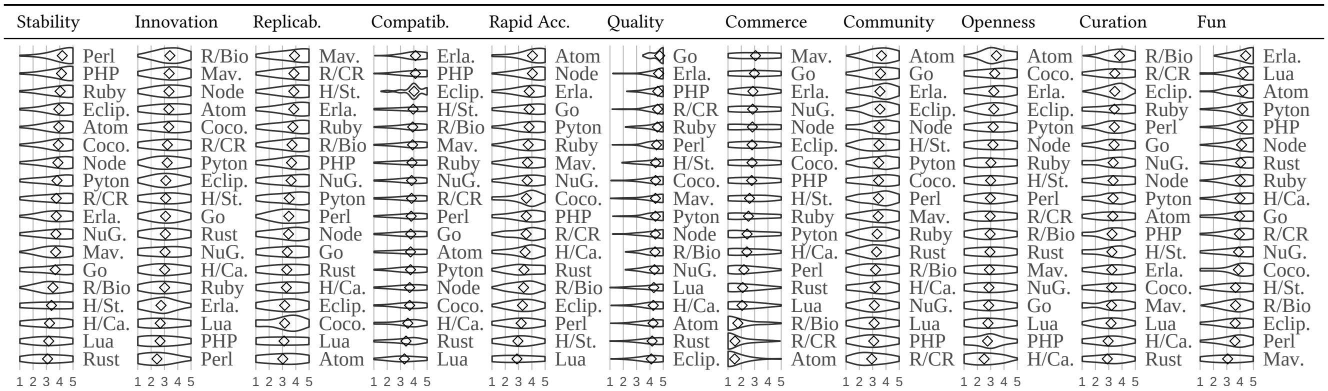
- When working on <package>, I must expend substantial effort to find versions of all my dependencies that will work together. — [agreement scale, as above]
- (OPTIONAL) Compare <ecosystem> with other ecosystems you've used or heard about – does one have some features that the other should adopt? If so, name the other ecosystem(s) and describe the feature(s). — [text field]
- (OPTIONAL) Why do you think people chose to design these other ecosystem(s) differently from <ecosystem>? — [text field]

Part V: Demographics and motivations.

- Age
  - 18-24
  - 25-34
  - 35-44
  - 45-54
  - 55-64
  - 65+
- Gender — [male/female/other]
- Formal computer science education/training
  - None
  - Coursework
  - Degree
- How many years have you been contributing to open source? (in any way, including writing code, documentation, engaging in discussions, etc) — [same time scale as "years used ecosystem" above]
- How many years have you been developing or maintaining software? — [same as previous]
- (OPTIONAL) Is there anything else we should have asked, that would help us better understand your experience with community values and breaking changes in <ecosystem> If so, tell us about it: — [text field]

B PLOTS OF ADDITIONAL RESULTS

Figure 4 plots personal values for each ecosystem.



1: not important or opposed, 2: somewhat important, 3: important, 4: very important, 5: extremely important; plots show smoothed distribution; diamond indicates mean

Figure 4: Personal values.