Noname manuscript No.
(will be inserted by the editor)

The Role of Surprisal in Issue Trackers

James Caddy · Christoph Treude · Markus Wagner · Earl T. Barr

Received: date / Accepted: date

Abstract Context: Software development creates and relies on a large volume of information, yet the volume of this information can make it challenging for developers to maintain an overview of all goings-on that a team and external actors contribute to a project. We posit that unexpected or "surprising" events could serve as important signposts amidst this information overload. These unexpected events may indicate underlying anomalies or emergent situations that require immediate attention. To explore this premise, our study leverages the concept of 'surprisal' from information theory to identify and quantify these unusual occurrences from the issues and pull requests of popular open-source software repositories.

Objective: Drawing from a previously published research protocol, our study investigates whether a correlation exists between the 'surprisal' of issues and their perceived importance or difficulty within software repositories.

Results: We performed a comprehensive analysis of approximately two million issues and pull requests, gathered from 1,270 repositories. Their 'surprisal' was then examined in relation to several indicative metrics of difficulty and perceived importance. Our results indicate only a weak correlation. This

J. Caddy
The University of Adelaide
E-mail: james.caddy@adelaide.edu.au

C. Treude
Singapore Management University, School of Computing and Information Systems
E-mail: ctreude@smu.edu.sg

M. Wagner
Monash University, Faculty of Information Technology
E-mail: markus.wagner@monash.edu

E. T. Barr
University College London, Department of Computer Science
E-mail: e.barr@ucl.ac.uk

outcome underscores the need for further research to devise more effective strategies for helping developers prioritise issues.

Keywords self-information · n-gram · GitHub issues

1 Introduction

Software development generates a vast stream of events, each of which must be acted on, if only to ignore or discard it. To manage this information flood, developers filter and prioritise in light of their understanding of their project's typical behaviour. These prioritisations are imperfect and provisional and can let critical events slip through. Further, a surprising event, even if unimportant and immediately ignored, can be valuable if it increases a developer's confidence in their current work plan by affirming their current priorities. Thus, we hypothesise that developers need to be aware of sufficiently unexpected or 'surprising' events, even those that turn out to be noise, as they may either affirm a developer's current course of action or trigger a developer to revise their priorities. This hypothesis is supported by related work which found many developers want to be notified of unusual or surprising events in their software repositories, e.g., when summarising project activity [58], notifying developers about unusual commits [40,24], and for the identification of malicious content [23].

In the broad area of statistical analysis it is crucial to be able to detect outliers. As Agarwal and Gupta so succinctly describe them, "an outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism" [2], which has implications for the reliability of the data and relationships that the data may represent. They describe a plethora of detection mechanisms, informed by quantitative metrics that fit perfectly with surprisal. Outlier detection through surprisal or any other method guides empirical software engineering as data quality informs reliability here too [51]. Miller and Myers espouse the use of outlier detection to highlight and fix possible errors in text-based and systematic tasks [44]—despite an admittedly small sample size—which could have its own use in software engineering tasks. Its use does not stop at human application, and Vignero and Demey present a system by which artificial agents may benefit from the same epistemic value that surprise has in human agents [60]. This invites surprisal analysis to be used in artificial intelligence and its own applications.

Consider the scenario of a developer who has been away from a project for several weeks, perhaps due to vacation or a different assignment. Upon returning, they face the task of catching up with the project's progress and understanding the major changes that took place during their absence. Conventional resources, such as release notes or version histories, might be available, but these can be overwhelming or of limited value to the user if they are not curated carefully and include even trivial changes [6], §4.2. Because surprising events deviate from the norm, they can bring issues to the fore that might otherwise be overlooked amidst more predictable changes. We propose tooling that sifts through the sea of changes to highlight the most unexpected or 'surprising' ones.

Surprisal presents an opportunity to inform the software development process for newcomers to a project and experienced maintainers alike. Newcomers have their own preconception of the software development process, either from

a lack of experience or experience gained elsewhere in their careers. Should surprising issues hide insidious difficulty, then, they are a poor learning experience for developers with no experience. Surprisal analysis can therefore steer new developers away from such issues, into a more gentle and productive introduction to a software repository. For experienced newcomers, cross-project transfer of knowledge concerning "surprises" would improve the engineering process if those surprises confer concerns or better methods to achieve their desired outcomes. For example, two projects, one that uses a third-party library for a feature, and the other that has a custom implementation, could learn the benefits of the other method through a newcomer's perspective of the other project where they have experience.

It is important to note that experienced developers have an entirely different perspective of surprising issues that a project may have. This comes from the extra dimension that the length of time maintaining a project represents. A particular issue may not be surprising to an experienced developer who expects particular unstable features to present problems in the future. This perspective aligns better with what we expect a dedicated statistical model would represent, compared to a newcomer, which could alternatively be represented with a cross-project model.

Surprisal, a concept derived from the foundational work of Claude Shannon in information theory, is a measure of how surprising an event is, given its probability under a random variable. It quantifies the informativeness of the content of that event, with higher surprisal indicating more unexpected or informative content. This work explores whether 'surprisal' can effectively help developers navigate the vast information terrain of software development. The formal definition of surprisal for an event x is given as $I(x) = -\log_2 P(x)$, where $P(x)$ is the probability of x occurring. This concept is frequently applied in the context of language models, where it is traditionally used as a proxy for the cognitive load of parsing a model's output [26].

In line with recent work applying natural language techniques to software engineering data [30], we train a language model on issues submitted to GitHub in this study. Our aim is to identify the most surprising ones and to assess the usefulness of surprisal in identifying issues that developers would want to be aware of. We explore two scenarios involving surprisal: its impact on resolution difficulty and the perceived importance of surprising issues. Although we analyze issues and pull requests separately in this study, we will refer to these collectively as 'issues' for convenience.

Previous research [35] suggested that adhering to project-specific language norms can reduce issue resolution time. With the assumption that surprising issues are likely to deviate from these norms, we investigate whether they present more difficulty in resolution. Other studies [45] have looked into the types of issues that get reopened, and found that specific issue metrics, such as the number of comments, show correlation. We conceptualize difficulty in terms of (1) reopened rate, (2) amount of discussion, and (3) resolution time. We propose that issues with lower surprisal are resolved faster due to less difficulty, while those with higher surprisal take longer.

Moreover, previous work asserts that developers wish to be alerted about unexpected issues in their repositories [58], and that issues of high importance are more likely to feature in release notes [1]. Surprising events are not necessarily important, but automatically prioritising them can fail, so each one needs to be assessed, which can trigger revising priorities. In this context, we explore whether surprising issues are quickly assessed or reorder priorities. We do so in terms of (1) mentions in release notes, (2) being the first issues tackled after a break, (3) attracting GitHub reactions [7], and (4) being assigned priority labels. We hypothesize that issues with lower surprisal are discussed less and therefore leave fewer traces in a project's development artefacts. In contrast, we propose that surprising issues—those that hold the most novelty—are among the most discussed events. Their resolutions are more frequently communicated to users; they are prioritised when developers sift through the backlog; and they garner interest not just from developers, but users as well.

In the interests of the scientific community, particularly in transparency and replicability, we have made the useful data and source code used in the paper available on Zenodo [10].

2 Research Questions

To guide our investigation, we have formulated several research questions and hypotheses, building on the research protocol detailed in our prior work [12]. The primary question we aim to answer is, "Is surprisal useful in identifying issues that developers would want to be aware of?". Given the absence of a clear metric for awareness needs, we turn to related research for proxy metrics which can be grouped into three categories: the human surprise emotion (addressed by RQ 1); resolution difficulty (addressed by RQs 2 and 3); and perceived importance (addressed by RQs 4 and 5).

> **RQ1**: How well does information theory's surprisal, as measured by a statistical language model, align with perceived surprisal?

With RQ1, we hope to find how statistical language models compare to the human perception of surprisal. The statistical language model will accurately calculate the likelihood of the tokens within an issue, but do so based on the millions of tokens. The best ability of humans is likely unable to achieve such speed and precision, but will be able to estimate a general level of surprisal based on a working-memory of the issues that they have been exposed to. This research question aims to answer how accurate this judgment could be, and if it is potentially helpful for ad-hoc surprisal-based triage.

> **RQ2**: Is there a correlation between the surprisal of issues and the difficulty of their resolution?

For RQ2, we define 'resolution difficulty' in terms of three factors: the frequency of issue reopening; the extent of discussion prior to issue resolution; and the time taken to resolve the issue. While this is not an exhaustive list, these indicators will serve as a starting point for assessing issue difficulty in this study. We hypothesize that issues with higher surprisal, potentially signaling a greater level of unexpected complexity, may prove more challenging to resolve, as evidenced by these factors.

We formalise RQ2 into the following hypotheses:

$H_{2.1}$: Surprising issues are more likely to be reopened.
$H0_{2.1}$: There is no significant difference in how often surprising issues are reopened, compared to unsurprising issues.

$H_{2.2}$: Surprising issues attract more discussion. (Number of people involved)
$H0_{2.2}$: There is no significant difference in how much discussion surprising issues draw, compared to unsurprising issues.

$H_{2.3}$: Surprising issues attract more discussion. (Number of interactions)
$H0_{2.3}$: There is no significant difference in how much discussion surprising issues draw, compared to unsurprising issues.

$H_{2.4}$: Surprising issues take longer to resolve.
$H0_{2.4}$: There is no significant difference in time to resolve surprising issues, compared to unsurprising issues.

$H_{2.5}$: Surprising issues are difficult, and difficult issues are best represented as some combination of reopen rate, amount of discussion, and time to resolve.
$H0_{2.5}$: Surprising issues are difficult, but difficulty is best represented as only one factor of reopen rate, amount of discussion, or time to resolve.

RQ3: Is there a correlation between the surprisal of pull requests and the difficulty of their resolution?

RQ3 adopts the same definition of 'resolution difficulty' as RQ2, but applies it to pull requests instead of issues. Pull requests are more constrained than issues, typically addressing a clearly identified need, and thus could provide a better use case for our central hypothesis.

The formal hypotheses for RQ2 are shaped similarly to those for RQ2, as pull requests and issues share functional similarities in this context. To maintain brevity, we have omitted the detailed text for these hypotheses ($H_{3.1}$ through $H_{3.5}$).

RQ4: Is there a correlation between the surprisal of issues and their perceived importance?

For RQ4, "perceived importance" is defined as a combination of several factors: mention in release notes; rapid addressing after a break period; garnering a high number of GitHub reactions; and the assignment of high-priority labels.

While this may not be an exhaustive list, it establishes a sufficient basis for our investigation.

We formalise RQ4 into the following hypotheses:

$H_{4.1}$: Surprising issues are more likely to be mentioned in release notes.
$H0_{4.1}$: There is no significant difference in how often surprising issues are mentioned in release notes, compared to unsurprising issues.

$H_{4.2}$: Surprising issues are addressed with priority over unsurprising issues after a hiatus.
$H0_{4.2}$: There is no significant difference in the time addressing post-hiatus surprising issues, compared to unsurprising issues.

$H_{4.3}$: Surprising issues attract more GitHub reactions.
$H0_{4.3}$: There is no significant difference in how many reactions a surprising issue receives, compared to unsurprising issues.

$H_{4.4}$: Surprising issues are more often labelled as high-priority issues.
$H0_{4.4}$: There is no significant difference in what priority surprising issues are labelled, compared to unsurprising issues.

In $H_{4.2}$, we define a "hiatus" as belonging to the top 25% of the longest periods between issue resolutions by each contributor for a specific repository. The term "priority" here refers to the sequence in which issues are tackled post-hiatus.

RQ5: Is there a correlation between the surprisal of pull requests and their perceived importance?

Just as RQ2 and RQ3 focus on issues while comparing them with pull requests, RQ4 and RQ5 adopt a similar approach. Pull requests are considered separately due to their more formal structure and the fact that they directly address needs typically expressed in issues. The complete formulation of hypotheses $H_{5.1}$ through $H_{5.4}$ has been excluded for brevity's sake.

3 Variables

A summary of the variables involved for all the research questions can be found in **??**.

Predictor Variable:

- **Surprisal.** How surprising the content of the issue is to the language model. This is calculated as the cross-entropy of the issue text and the corpus of issues. Cross entropy (Equation (1)), first described as 'minimum directed divergence' by S. Kullback [37], calculates how many symbols on average are required to represent an event from one distribution in a coding optimised for another, if both have the same support. In our setting, it measures how

many symbols are required to represent an issue's description, based on the distribution of words observed (P_o) in the issue, in the true distribution (P_{tt}) of words in all issues.

$$H(P_o, P_{tt}) = -\sum_{x \in X} P_o(x) \times \log(P_{tt}(x)) \tag{1}$$

When the observed distribution (the issue's use of words) differs from the distribution of the training set (the set of all issues), cross entropy increases. Higher values of cross entropy therefore imply an issue is more surprising.

Response Variables:

- **Surprise.** The novelty of information perceived by a human is one reason someone might say something is a "surprise" [63]. This emotional response is subjective and imprecise when compared to the machine-calculated "surprisal". For this reason, we use a 4-degree Likert scale comprising the ratings 'Very Unsurprising', 'Unsurprising', 'Surprising', and 'Very Surprising'. We chose to use four degrees to avoid a neutral option, which could be misused as a default option or crutch for lack of surety, which likely occurs as Chyung *et al.* explain [14].
- **Reopenings.** How many times the issue has been reopened. An issue can be labelled as closed and then reopened for numerous reasons, just as a pull request can be merged and then reopened. This usually indicates a regression of functionality, reoccuring bug, or unsuccessful fix [32], all indications that the issue has additional complexity associated with it.
- **Participants.** How many individual participants have interacted with the issue. Every event that takes place on an issue has an actor associated with it. This actor represents somebody interacting with the issue. If a particular issue involves multiple assignees for example, it may be a sign that additional expertise is needed to resolve it. It could also mean that it affects a lot of people but is not necessarily more difficult. Kavaler *et al.* [35] show that there is a significant increase in issue resolution time with an increased number of unique participants.
- **Interactions.** How many interactions have been made with the issue, including comments, mentions, taggings, assignments and state changes. A full list of events is available in the GitHub Issue API documentation [17], including those that have not been used. Some of these interactions are considered a normal part of the resolution process, although we expect to see more interactions if the issue reveals hidden complexity over time. Kavaler *et al.* [35] also show there is an increase in issue resolution time corresponding to the number of comments made.
- **Open State Duration.** How long the issue has been unresolved in seconds; from first submission to last closure, or if it has not been closed, the time of analysis. While some issues may not be difficult, but especially time-consuming, we expect to see longer resolution times for issues that are difficult to diagnose or replicate.

- **Mentions in Release Notes.** How many times the issue has been mentioned within release notes on GitHub. Some repositories make no use of GitHub's Releases feature, so only the repositories that do, and that mention issues at all in them will be considered. Highly important issues are more likely to be included in release notes [1].
- **Order of Address.** After lengthy breaks of development, whether independently taken or due to holidays, it is likely that the most pressing issues in the backlog are chosen for immediate resolution. Commits after extended breaks have been described as interesting in a previous paper [58]. Each contributor has a time between addressing issues, so taking the top 25% of these breaks in terms of longest duration, and then ordering the issues that they worked on afterwards gives us an indication of what importance the author places on each issue.
- **Reactions.** How many reactions have been made on the issue. Reactions give users a quick way to interact with an issue. For example, users can express joy that a particular issue is closed, or frustration if it disrupts them personally, through reactions. The number of reactions can be seen as a community rating of importance, rather than that of the maintainers [7].
- **Labelling.** Many repositories use the GitHub issue labelling system to triage incoming bug reports and feature requests. Issues are sorted by maintainers into perceived priorities and labelled as such, from low priority to high priority [64], §3.B. These labels can be seen as the maintainer's rating of importance.

4 Data Sets

In this section, we present the data sources that we used, and how we used them.

4.1 Sources

For this study, open-source software stored on GitHub serves as our primary and sole source for software issues. Using the GitHub REST API, we extract the raw text of the issues, their titles, descriptions, labels, etc. The data is current as of March 2023. We also extract timestamps, event data and numerical data, corresponding to the response variables. An example of the data that this interface provides can be found in the GitHub documentation [18]. Stars represent a user liking a repository, and therefore indicate popular repositories, more likely to have a high number of issues due to increased testing and feature requests — a result of more users [8]. To maximise the number of users that might benefit from this research, while limiting the time and computational resources to a manageable amount, we have decided to start with the top 5,000 most-starred repositories. The GitHub API does not provide direct access to

Table 1 Additional information on the variables present in the study, including how they are measured.

Variable	Hypotheses	Description
Surprisal	Predictor for all hypotheses	How surprising an issue is to the statistical language model.
Surprise	Response for $H_{1.1}$	Subjective human rating of the emotional surprise that an issue evokes.
Reopenings	Response for $H_{2.1}$, $H_{3.1}$	After an issue has been labelled as closed or resolved, it can be reopened either due to a fix being unsuccessful, a regression, or reoccurring bug.
Participants	Response for $H_{2.2}$, $H_{3.2}$	Number of unique individuals involved with the issue. Each event (as described by the GitHub API [17]) associated with an issue has an actor that initiates it, whether that event is a comment, or a state update.
Interactions	Response for $H_{2.3}$, $H_{3.3}$	Number of events associated with the issue.
Open State Duration	Response for $H_{2.4}$, $H_{3.4}$	Length of time between the issue's creation, and it being resolved for the final time in seconds.
Mentions	Response for $H_{4.1}$, $H_{5.1}$	Number of mentions within a repository's release notes.
Order of Address	Response for $H_{4.2}$, $H_{5.2}$	Issue order after longest 25% of contributor's inter-issue resolution times.
Reactions	Response for $H_{4.3}$, $H_{5.3}$	Number of reactions on a particular issue.
Labelling	Response for $H_{4.4}$, $H_{5.4}$	Perceived priority or importance label of a particular issue.

table]tab:variables

Variable	Measure	Operationalisation
Surprisal	Ratio	Cross entropy of issue, obtained with probability from SLM trained on corpus of all issues.
Surprise	Ordinal	Assessed by a human participant, by rating sample issues on a 4-degree Likert scale.
Reopenings	Ratio	GitHub Issue API's "reopened" event.
Participants	Ratio	GitHub Issue API's event "actor" for each event associated with an issue.
Interactions	Ratio	Count of events, as described by the GitHub Issue API [17].
Open State Duration	Ratio	Difference between GitHub Issue API's "created_at" value for the issue and "closed_at" value.
Mentions	Ratio	Scrape for issue numbers through GitHub's Releases API.
Order of Address	Interval	Issues assigned to a contributor are retrieved through the GitHub Issues API.
Reactions	Ratio	Count of reactions for an issue, from GitHub's Reactions API.
Labelling	Interval	Labels are extracted via the GitHub Issues API, and then normalised (per repository) to a 3-degree scale, 'low-importance', 'medium-importance', 'high-importance'.

such information, so we query the GitHub API with the filter that it does provide; returning all repositories with stars $\geq x$ for a low enough value of x to collect 5,000 results.

This supposes that GitHub is used in the same way by the maintainers of those 5,000 repositories, which is not the case. As noted by Kalliamvakou et al. in 2014 [33], many of GitHub's repositories do not make use of the Issues feature; 36.6% of these repositories are not software development related and possess little to no code [33]; and approximately 19,000 repositories ($< 0.1\%$) at the time were simply a mirror of a repository developed and hosted elsewhere. We use G-Repo [49] as a means to filter out non-software repositories and repositories making little use of the Issues feature (less than 100 issues). This is the final filtering step, where 1,270 repositories and their approximately 2,000,000 issues then go on to be included in the model.

Some of the repositories are not English language repositories, but there are few enough that they do not contribute a large proportion of the issues, only approximately 58,000 of 2,000,000 (as estimated by non-ASCII characters appearing in their text). As much as possible, we wish to limit the scope to English language issues, but precise detection of this proves challenging and thus we elect to leave these issues in the training data. Consequently, these issues appear as above average surprisal since their tokens do not appear in English language issues. Since the analysis based on ASCII encoding is only a very rough approximation and it is infeasible to judge all repositories with a language detection library, we did not attempt to exclude non-English language repositories. The corresponding threats to the validity of this analysis are discussed in Section 8.3.

4.2 Language Model Transfer

The statistical language model (SLM) used to determine the surprisal of an issue is intended to represent a probabilistic model of what the content of an issue looks like. Software issues are typically written in such a way that requires domain-specific knowledge of terminologies and jargon, and are in most cases very specific to the project they reference. As a result, a more specific SLM, trained on a software-based corpus is likely to see some improvement in accurately modelling the software language used, compared to a more general pre-trained model [61], §5.3. For this reason, a bespoke language model is trained for the task.

A subjective evaluation of the model takes place after it is trained. This is to ensure that the model is accurately representing the training data, and the probability that phrases occur are reflected in the count of the n-grams. This is a simple process where the n-grams are sorted by their occurrence count and manually inspected, considering several from the top and bottom of the range. N-grams that appear the most should be common phrases or structures found in issues, and the opposite being true for n-grams that appear the least. We expect to find that the n-grams "require bigger architecture" and

"languages interested having" would have low counts because of the dubious quality of the grammar, and we do, as they each have 1 count across all issues. Non-significant combinations of numbers, e.g. "8 10 110" appear only once, as do hexadecimal strings e.g. "6ae821421a7d downloading a9e976e3aa6d" and "code 1521 ea3859d4ba2f3e577a159bc91e3074c5d85c0523". These observations lend credence to the correct training of the model. Similarly, examples from the high count end of the range do too. For example, a very common structure parsed with 2 million n-gram occurrences, was nested HTML markup, simply "[HTML] [HTML] [HTML]". Given the structured nature of HTML, we should expect such n-grams to occur regularly. Greetings and valedictions should occur frequently at the start and end of issues respectively. The model also shows this with "[BLANK] [BLANK] hello" occurring 28,825 times, and "thanks [BLANK] [BLANK]" occurring 26,270 times.

N-gram models saw some popularity during the 2000s and 2010s, but gradually saw declining popularity due to their perceived contextual fragility [50](2000), and competitive neural language models being developed [43](2012). Despite this, n-gram models still perform admirably in Natural Language Processing applications. For our SLM, we use a trigram model using word tokens (shingles). We chose this due to its simplicity, familiarity, popularity, and reasonably effective results [13].

4.3 Pre-processing

Issues are composed of a title and description. Both of these elements have the possibility of individually containing pertinent information, and in some examples do not describe the information the other holds. It is for this reason that during the pre-processing stage, we prepend the issue description with the issue title. The SLM is trained on these title-description combinations.

Before training a model on the issue text, we will clean the data with the following pre-processing steps:

1. HTML Void Elements [57] are translated into special tokens, e.g., `<br>` becomes `[BR]`.
   (Also code blocks, see details following.)
2. Other HTML elements are replaced with their contents, e.g., a list element becomes a simple string of its content.
3. Text undergoes normalisation of its Unicode forms. Canonical Composition (NFC) is used in accordance with the Character Model standard proposed by W3C [25].
4. Punctuation and symbols are removed on word boundaries, and when isolated.
5. Stop words are removed.
   (See details following.)

As singular code tokens — variable names and the like — appearing inline in the body text of the description may provide important contextual information

across multiple issues, they will be preserved in the training data if they are not formatted similarly to code blocks. Code blocks, on the other hand, risk introducing too many globally-unique tokens into the model without introducing useful and actionable information to the description. The reason is that the syntax of code can be entirely disparate with that of English, for example, and so code blocks will be replaced with a special token `[CODE]` where they are demarcated with the `<pre>` and `<code>` HTML tags (as is typical on GitHub). Unfortunately, due to the varying quality of user contributions to GitHub, it is impossible to detect all the code that appears in issues. While many people use the correct formatting for code, others are unaware of this convention. While most code blocks have been replaced with a special token, many examples of code will slip through into the language model. These code blocks are easily detectable via manual inspection, but not in a timely, automated way across many millions of issues, and as a consequence will artificially raise the surprisal of an issue. The assumption that we have made is that code blocks either appear as the markdown ``` or HTML `<code>` tags wrapping a block of text. Singular code tokens that appear this way will also be removed to simplify processing.

As is standard in many natural language applications, we perform stop word removal using the set of stop words provided by the natural language toolkit spaCy. Stop words are the most common words in the language, and "help build ideas but do not carry any significance themselves" [41]. Removing these removes low-information tokens from the documents and increases average entropy. This also contributes to lowering the processing time and memory required as there are fewer words to track, and thus increases the efficiency of the model. It is possible that the quality of the language model could be improved by further transforming the text in a final step. In other applications, lemmatisation or stemming is used to reduce the occurrence of high-information tokens. Abstract concepts are learned better if "database" and "datastream" are reduced to the concept "data". In the research protocol, we state that a trial of this will be conducted to see if it is suitable for models measuring surprisal. Unfortunately, due to lack of time and effective tooling, stemming was not used, so the model, while accurate to the training data, loses some of its transferability to other data sets.

An overview of the steps taken to prepare the data and train the statistical language model can be found in **??**.

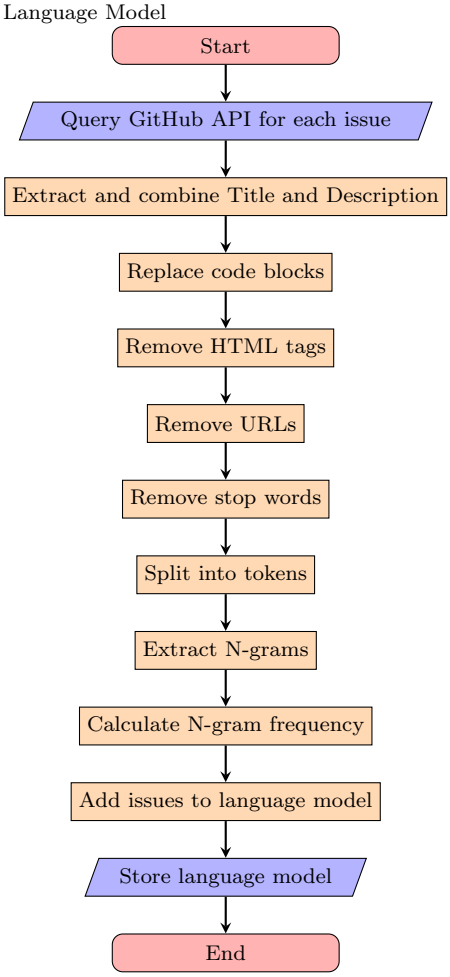
5 Execution

In this section, we present the method we use for the study.

5.1 Method

To test all hypotheses, we adopted the following method, as briefly outlined in **??**:

Fig. 1 Training the Statistical Language Model



figure]fig:training

We gathered data by querying the 1,270 repositories that we selected from the most starred repositories on GitHub using the GitHub API. The API provided us with the ID of each issue that had been created within the repository. We used this ID to interrogate the API for each issue of the repositories. The information returned to us included each record kept within the issues. We isolated the records for the titles and descriptions, preparing them for the next step.

We put the titles and descriptions through the pre-processing steps outlined in the previous section. At the conclusion of the pre-processing step, we fed the tokens into the statistical language model's initial learning process. We took the context of each token into account, tallied n-grams, and calculated the relative n-gram frequencies. Once we had trained the SLM, we considered the rest of the information from each issue individually.

We examined the list of labels associated with each issue. We recorded issues that had labels relating to priority or importance as having them, and we normalised the highest priority/importance label in the list for an issue into the range of three degrees. For more detail on the normalisation process, please refer to Section 5.3 that follows.

We used the title and description again to measure the resulting surprisal of the issue. We interrogated each n-gram from the concatenated title and description in the SLM for its surprisal value, and we recorded the average of each n-gram's surprisal to represent the overall surprisal of the issue.

We checked the list of events associated with an issue, which contained multiple data points that we recorded for the correlations. We tallied each event as the number of interactions, we also tallied each event marking a reopening, and we counted the set of unique authors for the events.

We noted the open duration as the difference in seconds from the opening of the issue to its recorded closing. Like GitHub, we only took into account the most recent closing. For each issue, we also separately logged the user that resolved it along with the closing time. This information aided us in determining the inter-issue resolution time for each issue later.

Each reaction type for an issue has a separate count of how many users have made that reaction. We record the total number of reactions to the issue, aggregating all reaction types; "+1", "-1", "laugh", "confused", "heart", "hooray", "rocket", and "eyes" into the one total.

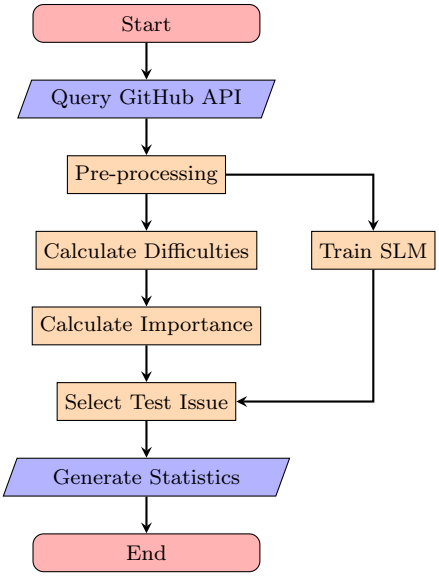
We determined whether or not an issue was also a pull request from the presence of the "pull-request" field in the issue's returned information. Of all the issues we analyzed, approximately 1.05 million (53%) were pull requests.

That marked the end of the information that we could extract by-issue from the GitHub API. For the remaining data, we queried each repository's release notes using the GitHub API. In release notes, it is common practice to cross-reference which issues were resolved in a particular release. Release note authors only need to mention the issue's number for GitHub to create a cross-reference. As we parsed the release notes, we tallied every mention of a particular issue's number in the repository's release history and recorded it with the issue.

After the step in which we had recorded the closing user and resolution time for each issue, we calculated the longest 25% of inter-issue resolution times per unique contributor. We partitioned the chronological list of issues that a contributor had resolved into bins, splitting when the time between issues was within the durations of those longest 25% of breaks. We recorded each issue's ordinal position in the bin. As a result, the issue immediately following a break determined to be within 25% of the longest breaks received the rank of 1.

We completed the processing of data and all that remained was to generate descriptive and inferential statistics as per Section 6. We calculated the mean, standard deviation, minimum and maximum values for each variable, and generated a plot showing the relationship between each variable and surprisal. Finally, we ran Kendall correlations for each response variable.

Fig. 2 Method Flowchart



figure]fig:method

5.2 Human Surprise Rating

To judge the applicability of machine rating of surprisal as an analog for human surprise, it is necessary to have a human participant rate a sampling of issues by their relative 'surprise' and calculate the machine rating of the surprisal of those same issues. For manual analysis, we mark 328 randomly selected issues, a sample size calculated to attain a 95% confidence interval, from an arbitrary but representative repository totalling 2,887 issues, which is the approximate mean number of issues per repository from the entire GitHub data set.

We task two authors of this paper to rate all 328 issues by how surprising they are. The second rater is only required to determine the subjectivity of the task, by measuring inter-rater agreement, they will not contribute to the comparison with the machine rating. Each rater is required to read all issues in advance before rating them, to gain an understanding of what a typical issue looks like in the repository. They are only asked to take note of and rate the title and body text of the issue, and no accompanying information such as labels, issue type, responses, images, etc. The participant is asked to rate each issue on a 4-degree Likert scale with the possible options of 'Very Unsurprising', 'Unsurprising', 'Surprising', and 'Very Surprising', according to how surprising the content of the issue is to see in the samples. An example of a very unsurprising issue might be formatted in the repository's standard bug report form and describe a bug that has either occurred before, or is similar to one that has. A surprising issue counterexample might consist of unstructured text, describing an issue that has no similar reported issues, perhaps even

describing an issue not caused by the software repository in question but external software instead.

As mentioned, the raters and model only take the title and body text of an issue into account. Additional information from the issues such as images were not considered in the human rating of surprisal as we were unable to devise a solution that would expose the statistical language model to this data too. The intention of this experiment is to see how well surprise aligns with surprisal, which means that it is important that both sides only have access to the same information. The consequence of this choice, is that the human raters cannot give a truly accurate judgement of an issue's total surprise. While this allows direct comparison between human and machine, we also chose to limit the other textual data exposed to both sides to simplify the process, and it is the omission of labels, responses, etc. that may limit the reliability of the conclusions presented.

It is assumed that the 328 issues that the human sees give an indicative view of the entire repository. The statistical language model used to calculate the surprisal of these issues is afforded more information; the entire set of issues from the repository. This was a compromise made to lower the amount of tedious work and reduce fatigue-induced error on the behalf of the human rater, but ensure that the language model had enough data to construct a dense enough language model that would accurately rate the surprisal. All issues undergo the applicable pre-processing steps before they are consumed to create the statistical language model, after which the model is queried for the surprisal values of those issues specifically selected for human rating.

The analysis is carried out in two parts. Firstly, the rater agreement is calculated between both human and the machine ratings. A well-suited measure for Likert scales, and ordinal measurement levels is Krippendorff's alpha coefficient [28], as it takes into account the relative severity of disagreements between distant rating values. Secondly, a correlation is calculated between the human and machine ratings. We use the ratings given by the first author, as either is valid to use, since agreement is moderate as shown in Table 6 but not problematically low. A description of the tests performed is found in the following Section 6, Analysis. The difference between 'Very unsurprising' and 'Unsurprising' and similarly for surprising issues may not prove to be useful, so we additionally include the analysis of the binary case, where issues are only classified as unsurprising or surprising, without the degree of severity.

5.3 Label Normalisation

Some GitHub repositories use the labelling system to assign priority grades or importance to issues in their triage process. These labels are user text input and can represent these grades in a number of different ways. For example, one repository may use the labels 'Low Priority', and 'High Priority', where another may use the labels 'P1' through 'P5'. This labelling is specific to a

particular project, and represents the developers' best understanding of what order that issues should be resolved in.

To process these priority labels, it is necessary for manual classification to normalise these different ranges. Taking a random sample of 2,000 unique labels from 29,168 total observed labels, we task three of the authors to categorise each label as either not related to priority, or representing a priority from low, medium, or high. The agreement between these sets of ratings is then calculated using Krippendorf's alpha with a resulting value of 0.681. Higher is better; however, this value shows that there is a good consensus between the sets [36]. One of the authors is then able to rank all 29,168 labels knowing that those judgements are likely reasonable. More detailed information on this process has been published in the supplemental paper *Prioritising GitHub Priority Labels* which is available as a pre-print on arXiv [11].

These categories are normalised to the numerical range of -1.0 for low priority, 0.0 for medium priority, and 1.0 for high priority for the purpose of the analysis.

6 Analysis

This is a correlational study of independent random variables, the following is the design of the quantitative analysis that is undertaken. All calculations are performed using the JASP statistics program and the Python package SciPy. The following outlines the structure of the results section:

Descriptive Statistics We present descriptive statistics of the predictor and response variables in a summary of the complete data set. Sample size, mean, standard deviation, maximum, and minimum values for each variable are included.

Inferential Statistics We present a correlation and regression analysis using linear regression. We then test each hypothesis separately with its corresponding variable, and look for statistical significance.

Typically, a Shapiro-Wilk test is used to determine whether the data shows normality [54]. If the data is normally distributed, we can choose to use the additional descriptive power of a Pearson correlation. If not, a Spearman correlation can be used instead [56], §1.2.a). Because the sample size of issues is large, the Shapiro-Wilk test becomes too sensitive to deviations from the perfect model [65]. The Pearson correlation does not lose all accuracy with a non-normal population distribution, but can form the best estimate when the population has this property. The Kendall rank correlation is also useful to use, as it compares concordant and discordant pairs, unlike the Spearman correlation, which is purely based on rank. Given that the data is likely to have many tied ranks, since some response values are only integers within a very small range, it may be more accurate to use the Kendall correlation coefficient.

To test $H_{1.5}$ and $H_{2.5}$, we perform a multiple linear regression, this time including one statistically significant measure of difficulty into the null model. If the F-test is **not** statistically significant, we can accept the alternate hypothesis.

This test shows if a combination of our chosen difficulty factors is better in representing difficulty as a whole, in comparison to the measure added to the null model on its own. When performing this second regression, we expect to see multicollinearity, a high Variance Inflation Factor for these measures [39], §10.5, as our belief is that they represent the same variable: difficulty.

7 Results

Table 2 Descriptive statistics showing a general overview of the data collected for the predictor and response variables.

table]tab:descriptiveStatistics

	Mean	Std. Deviation	Minimum	Maximum
surprisal	22.801	3.720	9.825	26.610
reopenings	0.044	0.242	0.000	11.000
unique_participants	2.649	2.418	0.000	117.000
interactions	8.715	15.572	0.000	5913.000
open_duration (s)	5.516×10^6	1.811×10^7	0.000	3.272×10^8
reactions	0.195	2.216	0.000	497.000
label_priority	−0.241	0.920	−1.000	1.000
mentions_in_releases	0.105	0.636	0.000	92.000

See **??** for the descriptive statistics of surprisal and the response variables, see **??** for the correlation results, and see **??** and **??** for the linear regression results. Below, we discuss these results and the scatter plots of the data with relation to the research questions posed.

RQ1: How well does information theory's surprisal, as measured by a statistical language model, align with perceived surprisal?

A: Surprisal judgement differs substantially even between humans. Statistical models differ even more, with even less correlation.

Hypothesis 1.1 In Table 6 and Table 7 we see that both 'human versus human' and 'human versus machine' ratings show little agreement with how surprising the issues are. Given that the agreement between the human ratings is moderate but not low, we use the ratings of the first author when comparing against the machine ratings, since the difference is not too great. There is evidently more agreement between humans, including noticeably more in the binary decision, however, this is informed by a much smaller sample size of 50 out of

Table 3 Correlation values from the Kendall Rank Correlation test, between the predictor variable and each of the response variables, differentiating Issues and Pull Requests separately to show the minor differences.

All Issues	Kendall	
	tau B	p
reopenings	−0.0355	1.50e−129
unique_participants	−0.0696	0.00
interactions	−0.0801	0.00
open_duration	0.0333	4.07e−169
reactions	0.0198	2.41e−42
label_priority	−0.0678	1.01e−13
mentions_in_releases	0.0196	4.83e−41

table]tab:correlationTable

Only Pull Requests	Kendall	
	tau B	p
reopenings	−0.0166	9.61e−17
unique_participants	0.0172	1.85e−21
interactions	−0.0005	7.41e−1
open_duration	0.0649	0.00
reactions	0.0514	2.24e−148
resolution_priority	−0.0696	0.0
label_priority	0.1607	2.73e−7
mentions_in_releases	0.0181	4.99e−20

Table 4 ANOVA values from the linear regression, principally used for the F-test result showing the high independence of response variables.

table]tab:regression

Model		Sum of Squares	df	Mean Square	F	p
H_1	Regression	21768.255	4	5442.064	395.294	$< .001$
	Residual	4.245e6	308370	13.767		
	Total	4.267e6	308374			

328 samples. The correlations show similar disagreement in Table 8 where the 'human versus human' comparison shows a stronger positive correlation than that of the 'human versus machine' comparison.

RQ2: Is there a correlation between the surprisal of issues and the difficulty of their resolution?

A: From the analysis, we have observed that there is no correlation between surprisal and difficulty. Not according to the metrics used, or how we have decided to measure difficulty.

Table 5 Collinearity coefficient estimation results from the linear regression, showing low VIF (near 1) values displaying a low dependence between response variables.

table]tab:coefficients

							Collinearity Statistics	
Model		*b*	Std. Err.	*b**	t	p	Tolerance	VIF
H_0	(Intercept)	22.801	0.007		3403.760	< .001		
H_1	(Intercept)	22.871	0.010		2274.194	< .001		
	reopenings	−0.573	0.028	−0.038	−20.748	< .001	0.973	1.028
	unique_participants	−0.036	0.003	−0.023	−11.531	< .001	0.782	1.279
	interactions	−0.002	4.757e−4	−0.007	−3.396	< .001	0.786	1.273
	open_duration	1.159e−8	3.662e−10	0.057	31.650	< .001	0.987	1.013

b: unstandardised regression coefficient
b: standardised regression coefficient*

Table 6 Rater agreement coefficients of 'human versus human' ratings.

				95% CI	
Reliability Test	Degree	Coefficient	SE	Lower	Upper
Cohen's Unweighted Kappa	4	0.310	0.105	0.105	0.515
	binary	0.582	0.130	0.327	0.836
Fleiss' Kappa Overall	4	0.306	0.091	0.127	0.485
	binary	0.578	0.144	0.295	0.861
Krippendorff's Alpha	4	0.313	0.109	0.091	0.508
	binary	0.582	0.139	0.286	0.819

Table 7 Rater agreement coefficients of 'human versus machine' ratings.

				95% CI	
Reliability Test	Degree	Coefficient	SE	Lower	Upper
Cohen's Unweighted Kappa	4	−0.027	0.021	−0.068	0.015
	binary	0.023	0.038	−0.052	0.099
Fleiss' Kappa Overall	4	−0.185	0.038	−0.260	−0.110
	binary	−0.219	0.066	−0.347	−0.091
Krippendorff's Alpha	4	−0.360	0.053	−0.460	−0.254
	binary	−0.216	0.066	−0.348	−0.088

Table 8 Correlation coefficients in the comparison of 'human versus human' surprisal ratings and 'human versus machine' ratings. Both show relatively low correlation across the board.

					Kendall	
Ratings			Degrees	n	tau B	p
human	-	human	4	50	0.432	< .001
human	-	human	binary	50	0.595	< .001
machine	-	human	4	328	0.168	< .001
machine	-	human	binary	328	−0.117	0.029

Hypothesis 2.1 By comparing the surprisal value of an issue's description with the number of re-openings of the issue, we find that there is no correlation. By looking at the scatter plot of the results (**??**), it can be seen that there are signs that the hypothesis could be true, as we see a rightward skew where higher surprisal values do correspond to more re-openings. For example, we see that issues with two or three re-openings are much more likely to be in the top half of surprisal values. These glimpses of the hypothesis are, however, entirely dominated by the sheer volume of issues with zero to one re-openings that are spread evenly through the spectrum of surprisal values.

Hypothesis 2.2 By comparing the surprisal value of an issue's description with the number of unique participants of the issue, we find that there is a very weak negative correlation. In the scatter plot (**??**) we can also see the artifacts resultant of the training data. Repositories that ask users to submit their issues following a project-specified issue template; repositories that are less popular or have less activity have made an outlying spur on the left side of the graph. This dense area could indicate that issues following templates, by nature less surprising, may get resolved with less discourse because they are better defined in the first place. To confirm this, it would be valuable to look at a combination of number of unique participants and open duration for closed issues only, accounting for relative popularities of the repositories.

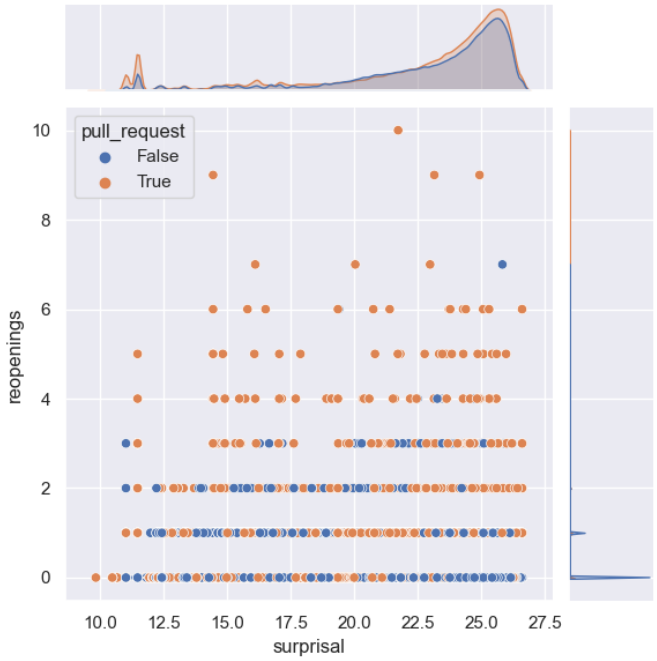
Hypothesis 2.3 When comparing the surprisal value of an issue's description with the number of total participant interactions with an issue, we find that there is a negligible negative correlation. Similarly to the correlation of unique participants, we can see artifacts of the training data in **??**. Participants are less likely to interact with the least and most surprising issues, and this could be for many reasons. Less surprising issues may not warrant much effort to fix, and highly surprising issues may actually dissuade potential participants from becoming involved.

Hypothesis 2.4 When comparing the surprisal value of an issue's description with the time taken to resolve an issue, specifically only within the context of the repository where that issue is from, we find that there is a very weak positive correlation, if any at all. This time, in **??**, we can clearly see some defined lines of similar and templated issues of low surprisal. However, the density of the figure obscures any patterns emerging in the higher surprisal ranges.

Hypothesis 2.5 By undertaking a multiple linear regression with the four previous response variables, we find that there is no indication that one of the variables wholly or significantly contributes to the difficulty of an issue,

as measured by its surprisal. This is confounded by the fact that no variables seem to contribute much at all. As seen in the output of the linear regression in **??** and **??**, we can see that the F-test shows that the means of the response variables significantly differ and we can reject the hypothesis that they are equal, and the Variance Inflation Factor is quite low, showing that each of these variables has an independent effect on surprisal.

Fig. 3 Number of times an issue is reopened. The majority of issues are never reopened.



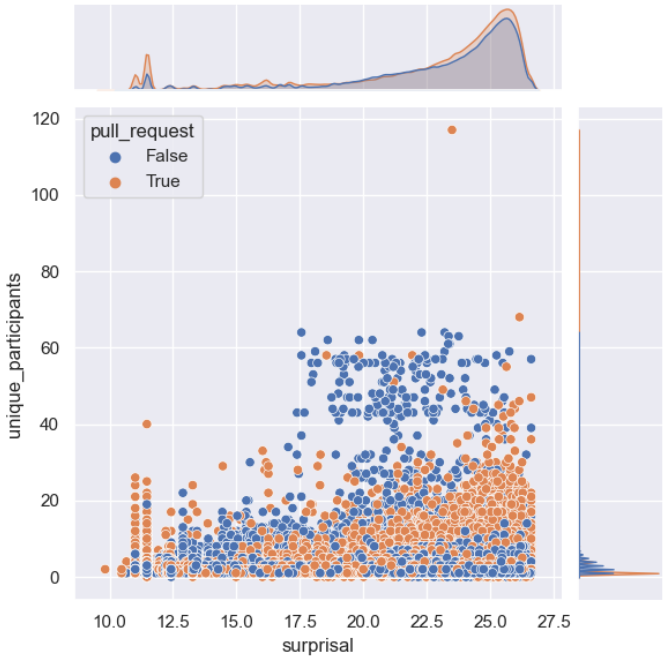
figure]fig:reopenings

RQ3: Is there a correlation between the surprisal of pull requests and the difficulty of their resolution?

A: We have observed no correlation between surprisal of pull requests and difficulty. The data does not support any difference of correlation between surprisal and difficulty for pull requests when compared to issues. See RQ2 above.

Hypotheses 3.1–3.5 Despite the difference of information contained between issues and pull requests, there is no evidence to suggest a deviation from the same conclusions. There is little to no difference in the correlations when we separate pull requests from all issues. For hypotheses 3.2 and 3.3, concerning

Fig. 4 Number of unique participants recorded interacting with an issue, showing distinct groupings within repositories.



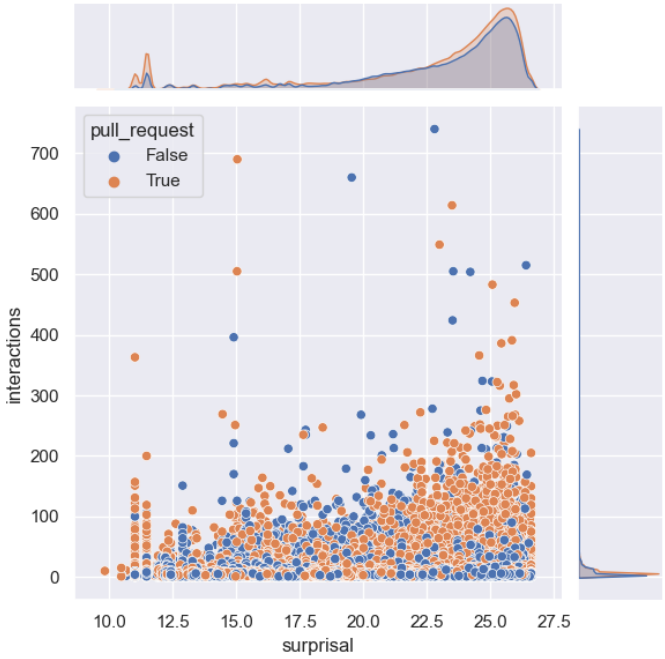
figure]fig:unique$_p$*articipants*

the amount of discussion with unique participants and interactions, we can see that there is an inversion of the correlation. Both the value for all issues and the value for pull requests only are so small, less than 10^{-20}, the difference is truly negligible.

> **RQ4**: Is there a correlation between the surprisal of issues and their perceived importance?
>
> *A: There is little evidence to suggest that surprisal is correlated with the perceived importance of issues. Not for how we defined perceived importance, evidently.*

Hypothesis 4.1 By comparing the surprisal value of an issue's description with how many times that issue is referred to by issue number in release notes, we find that there is essentially no correlation. This can be seen in **??**. In total, less than 6% of issues are ever mentioned in release notes, from quantitative observation this is likely because not many repositories use release notes as a way to document per-issue changes, but instead summarise or entirely neglect to mention the issues changed.

Fig. 5 Number of interactions from any user. There is one outlier value omitted of 5913



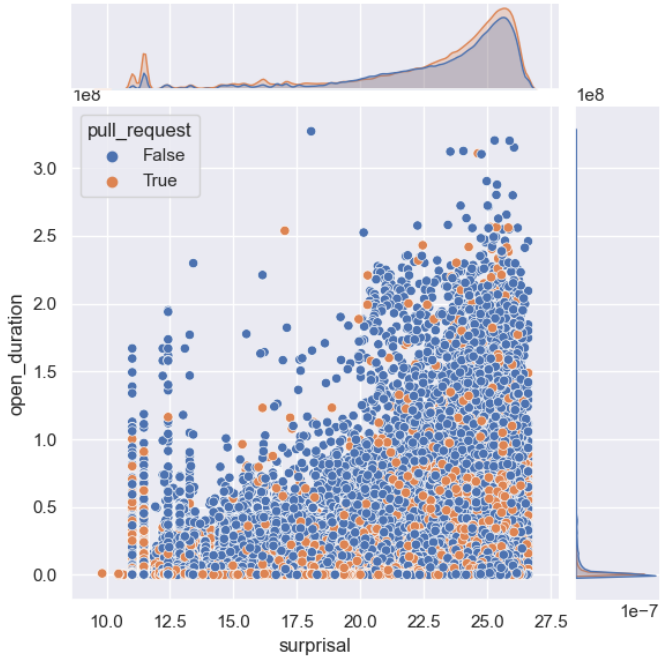
figure]fig:interactions

Hypothesis 4.2 By comparing the surprisal value of an issue's description with the order with which the issue is resolved by the resolver, we find that there is once again very little evidence of a correlation. In **??** we do see a clustering of high-surprisal issues being resolved first (rank 0), but equally, many high-surprisal issues are resolved with very little priority. The distribution is spread greater amongst the whole graph, though, showing that surprisal does not have much impact on resolution priority.

Hypothesis 4.3 When comparing the surprisal value of an issue's description with the number of GitHub reactions that the issue gathers, we find that there is no correlation. In total, less than 7% of issues have any reactions. It could be that the feature is not useful to users, as the feature is as old as 2016. The scatter plot (**??**) shows this overwhelming majority of issues distributed across the x-axis.

Hypothesis 4.4 When comparing the surprisal value of an issue's description with the labels that the issue gathers, we find that there is no correlation either. We also find that less than 2% of issues have labels pertaining to their perceived priority or importance. In **??** below, high priority labelled issues are

Fig. 6 Duration that an issue is open, from creation to final closing, in seconds.



figure]fig:open$_duration$

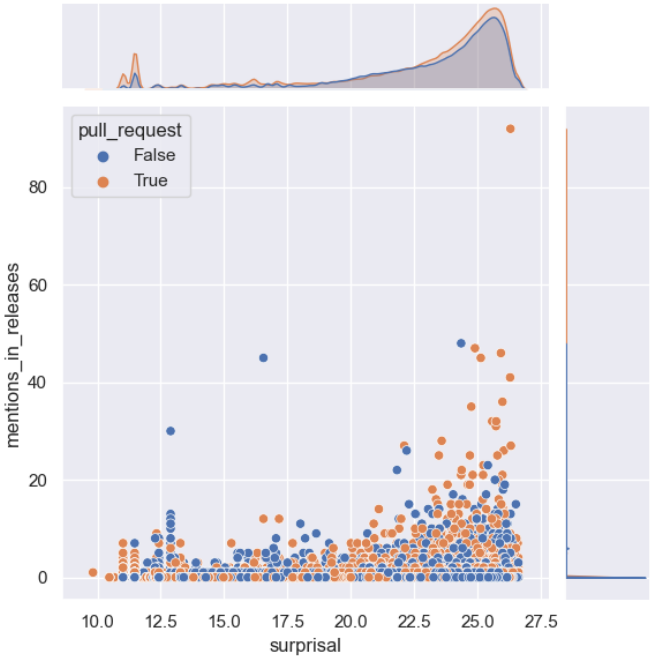
represented as having a *label_priority* value of 1.0, medium priority are 0.0, and low priority are −1.0.

RQ5: Is there a correlation between the surprisal of pull requests and their perceived importance?

A: There is little evidence to suggest that surprisal is correlated with the perceived importance of pull requests. The data does not support any difference of correlation between surprisal and perceived importance for pull requests when compared to issues. See RQ4 above.

Hypotheses 5.1–5.4 Despite the difference of information contained between issues and pull requests, we can be almost certain of the same conclusions. There is little to no difference in the correlations when we separate pull requests from all issues.

Fig. 7 Issues that gain mentions in release notes, and how many times they are mentioned.



figure]fig:mentions$_in_r eleases$

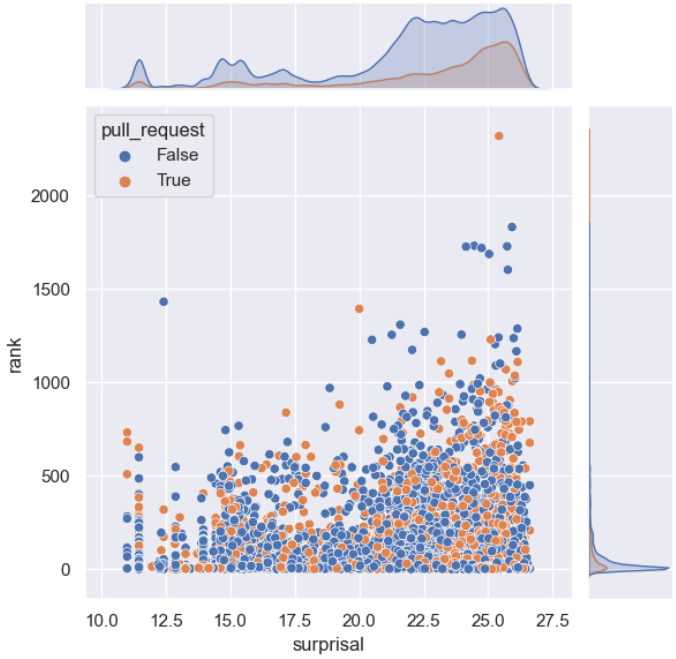
8 Discussion

In this section, we discuss the observed results, discuss deviations from the previously approved study protocol, as well as threats that could affect the validity of our work.

8.1 Perceived Surprisal

Research Question 1 concerns the link between the information theoretical measurement of surprisal, versus the human perception of surprisal. During the experiment we had the opportunity to compare sets of two human ratings of sample issues and the machine-calculated ratings. It is clear from the data, which shows very little agreement between any of the raters, that surprisal is a very subjective and potentially difficult metric with which to rate issues. Given that this is only based on a single repository, know that these results have a limited ability to generalise. To obtain a fairer judgement of perceived surprisal versus statistical surprisal, increasing the number of repositories in the experiment and asking the developers with in-depth knowledge of the repositories to rate the surprisal would increase the reliability of conclusions generated from the experiment.

Fig. 8 Resolution priority of issues after breaks. Developers with many resolved issues and therefore many breaks between issues can support higher ranked issues due to how many issues have been resolved.



figure]fig:resolution$_priority$

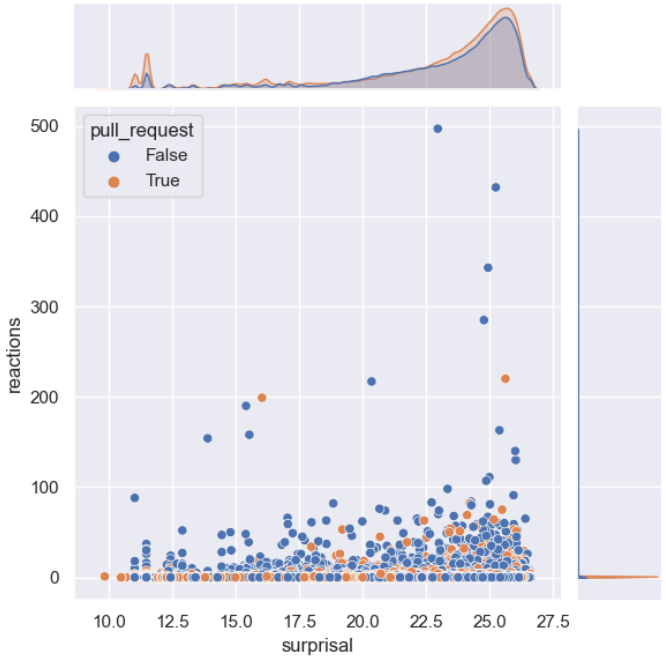
8.1.1 Human Agreement

The subjectivity was certainly shown when the initial ratings were discussed by the raters. Both raters had different ideas as to how the data should be approached, which types of information should be included in the rating, and how to organise the information. It was then necessary to debrief the raters on what disagreements arose, to ensure that the process was being followed correctly, and that disagreements were accurate.

An example of an issue which showed the most disagreement is shown in **??**. The disagreement between the raters is moderate, one rating it "surprising" and the other rating it "very unsurprising". When asked later, the first author explains a "surprising" level of surprise due to the concept of 'focus' not appearing elsewhere in the samples, and the inclusion of system information not found in other issues, notably the "A/B Experiments" section. The second author observes that the issue follows a very structured format, which appears commonly through the repository, and that the nature of the issue concerns an incidental problem to be expected from software citing *a posteriori* knowledge.

As the results section mentions, while the case of the 4-degree rating showed a lot of disagreement, there does seem to be more consensus on what is or

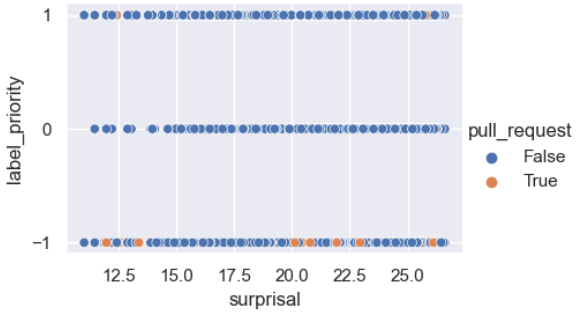
Fig. 9 Number of reactions each issue has received. Typically, issues receive 0 reactions.



figure]fig:reactions

Fig. 10 *Left:* Plot showing the corresponding numeric values for the normalised label priorities, where −1 is low-priority, up to 1, which is high priority. *Right:* Table showing the total number of issues counted that have the corresponding priority label.

figure]fig:label$_priority$



label_priority	issues
high	15772
medium	10791
low	32911

is not a surprising issue in the binary decision case. This may suggest that the fine measurement of surprise is difficult for humans, but the broad, coarse decision of whether it is—or is not surprising—is perhaps less subjective.

8.1.2 Model Agreement

An even greater difference in agreement occurs between the human and machine ratings of surprisal. We speculate that this happens for a couple of reasons discussed below.

Firstly, although the average surprisal of all words in an issue is considered during the evaluation of an issue, it is quite easy for the rating to be artificially inflated by the occurrence of a unique word, whose meaning or synonyms are much more common. This is most notable when the issue is short, and as a result, the overall average is more strongly affected by any outliers. An example of a relatively benign issue that is machine-rated as 'very surprising' is found in **??**.

Sanitising the text to extract tokens that are easily processed is a challenge without being able to account for all different ways that issue authors might lay their text out, their code, images or link references. It is almost certain that the text processing that was performed was unable to accurately provide 100% clean tokens to the model. In this case, some issues will show artificially inflated surprisal ratings too.

Secondly, this is a difficult task for humans, which is otherwise a straightforward calculation for a computer. To accurately rate whether an issue looks similar to 328 other issues, the participant is required to keep an idea of those 328 issues in memory. Familiarity with these issues can help with this, but increasing the number of issues to 10,000 which is below the maximum number of issues several repositories showed, and far below the total number of issues analysed, this becomes impossible. Reading and rating 328 issues also induce fatigue in human participants, which could definitely affect the accuracy of the ratings made as fatigue builds.

In summary, humans and machines look for different things when looking for surprisal, as evidenced by the disagreement between human ratings and approach to the task. If human participants have little agreement, the machine-calculated values might agree with one participant, but necessarily this does not agree with the other. The meaning behind the words is arguably more important than the words that are used, and while humans can easily process this compared to machines, understanding of words is far beyond the capabilities of this simple statistical language model. This suggests that a more robust model of the English language and a data representation to match may be able to increase the agreement between human and machine.

8.2 Deviations

Throughout the study, several deviations from the originally outlined protocol in the registered report were observed. While these alterations were primarily made for practical reasons, they were not subject to the peer review process. Below, we provide a comprehensive list of these deviations to ensure transparency:

- **Omission of Model Smoothing:** The original plan was to include a smoothing step in our statistical language model, with the goal of more accurately capturing the nuances of the GitHub issues corpus. We decided to not perform smoothing on the language model, which may result in model inaccuracies. However, as suggested by Chen and Goodman's comparison of smoothing techniques [13], larger models benefit less from smoothing. Given our relatively large training set, which contained no unknown tokens, the impact of this omission should be minimal.
- **Absence of Word Token Stemming:** To enhance the model's accuracy, we intended to implement word token stemming to abstract certain words to their core concepts. Unfortunately, this step was not carried out. We anticipate that this would have only marginally affected the surprisal values of many issues and would not have significantly influenced the observed correlations. Stemming typically reduces the surprisal values of issues, but due to time constraints and a lack of effective tools to handle the specific lexicon used in issues, we opted against it.
- **Inclusion of Non-English Repositories:** We initially decided to focus solely on English-language GitHub repositories to simplify the required data processing. This was intended to be performed using the G-Repo tool. However, due to a misunderstanding of its functionality, some non-English repositories were included in the data set. This could potentially inflate their surprisal values, as they are uncommon in a predominantly English data set.
- **Inappropriateness of Shapiro-Wilk Test:** The decision to use the Shapiro-Wilk test for checking data normality has been reassessed, given its known limitations for sample sizes greater than 5,000 [65].

8.3 Threats to Validity

This section outlines potential threats to the validity of our study. The external threats primarily include sample size and lack of model smoothing. The number of repositories and issues analysed, which is considered as the sample size, may pose a challenge to the validity of the experiment. With greater time and resources to spend, this limit could be increased.

Threats to the construct validity of our work encompass proxies for difficulty and perceived importance, repeatability, and reaction feature limitation. The measures used for difficulty and perceived importance were the best substitutes we considered, but this does not imply they were the optimal choices. These measures likely capture only some aspects of difficulty and perceived importance. Unseen influences associated with the chosen response variables could exist. Therefore, we represent the correlations as a general indication of difficulty and perceived importance, agreed upon by the researchers and independent reviewers. This is one assumption made in our experimental design. There are several reasons beyond surprisal that influence each of the response variables. These may dominate the effect of any influence that surprisal has. Assessing

correlation is most accurately achieved when the variables are independent of one another and unburdened by confounding factors. Without in-depth knowledge of how each repository's development is undertaken, it is impossible to fully know the extent of these, but some to consider include the following: some repositories will open new issues instead of reopening old issues; issue open duration varies widely due to non-development reasons; some repositories mention all issues resolved in their release notes in contrast to those that mention none; and order of address is largely dependent on the developers who may just instead pick the quickest or easiest issues all the time.

As much as the data used in the study affects the results, the data excluded might do so too. Particularly in the data gathering and pre-processing steps, potentially significant pieces of an issue can be lost. Of the information removed, the two most important pieces are the code snippets and images present in an issue. Both have the potential to represent information that is harder to represent or rarely represented in natural text. The reason for removing these is that it would be infeasible to represent this data in a way the SLM is able to interpret or provide useful calculations of surprisal for. Its exclusion represents information and therefore potential surprisal lost.

The accuracy with which the authors are able to rate the surprisal of issues within a repository is questionable without further research. While both raters felt confident doing so after reading through a sample of issues, a developer with more intimate knowledge of the entire set of issues from a repository may achieve better accuracy. A study involving the developers of repositories with several developers would be able to better inform whether this is the case or not. If developers are more accurate raters of surprisal, it would give an indication of the limits of human ratings of surprisal versus computed ratings.

Additionally, our study is impacted by repeatability concerns. GitHub is a dynamic data source. While some data types, such as git commits or issue events, can be filtered by time, others, like reactions and repository deletions, cannot. The implementation developed for extracting the response variable values was not coded with a time filter. This means that results could vary slightly or even significantly, depending on when the experiment is run.

Lastly, the reaction feature on GitHub was introduced in March 2016. Approximately 460,000 out of the around 2,000,000 issues in our data set were created before this date. Although these issues could receive reactions post-introduction, it does limit the total number of samples contributing to the analysis of reaction correlation.

9 Related Work

Shannon entropy, the basis for surprisal and other information science analysis, was first conceived in Claude Shannon's seminal work on information theory describing radio signals and their ability to communicate information [53]. Since then it has influenced many people and work in fields abroad of mathematics. One of Shannon's contemporaries, R. Fano shared Shannon's interest in this field

and wrote his own papers extending the field further. Cross entropy, as defined by Fano [19] currently dominates usage in the evaluation and optimisation of many language models, some large language models (LLMs), some statistical language models, but mostly neural language models. These problems are usually structured as optimisation problems, and traditional optimisation will use cross-entropy methods too [16].

Surprisal, a component of cross-entropy, has had a large impact in the domain of natural language. Specifically in psycho-linguistics where surprisal has predicted cognitive load [27], comprehension [42], and reading speed [20]. Analysis is typically the use case for surprisal even outside of natural language, but it has also seen use in novel generative works such as pun-generation [29, 34].

This study was directly inspired by the recent work on applying natural language techniques to software engineering data [30]. The study from Hindle *et. al.* presents a direction in applying techniques used in natural language to other constructed languages such as program code, whilst evaluating cross-project and cross-domain application.

Some defects in software can be seen as a direct consequence of difficulty. Automated defect prediction is a potentially very useful tool in software development that could save time and money for software companies. One paper directly inspired by the same study from Hindle *et. al.* was that of Wang *et. al.* which looked at applying surprisal, measured by a statistical language model (n-grams), to source code to detect defects [62]. Another study looking at defect prediction using n-grams that may have had independent inspiration was that of Y. Pang, X. Xue, and A.S. Namin [47]. As a summary of the work that followed this line of inquiry, a comprehensive analysis of surprisal as a defect-predictor was conducted by Ray *et. al.* [48], and a field summary has been provided by Allamanis *et. al.* [3].

Several authors have seen a problem in the current open-source software development landscape, where issue reports are bombarding developers who often cannot keep up and must resort to prioritising and backlogging tasks until they can address them. A large focus of software engineering research is going towards automatically prioritising issues, as the systematic literature review of Pone *et. al.* [9] argues. Many of the studies doing so use natural language processing as a means of prediction. One study using nine machine learning models is that of Shafiq *et. al.* [52], and studies using sentiment analysis instead are numerous [59,31,55]. The motivations present in our paper, of assisting developers overwhelmed with issues, are shared by other authors. There are several that have decided that new tooling is the way forward, and to increase "situational awareness" they create views of the software development from the information present in issue trackers [4,5,15].

There is one paper [38] from Kumari and Singh that shares many of the theoretical aspects and motivations as our paper. Both papers look at transforming the information mined from issue trackers from an entropy standpoint, and although the end products are different, they both intend to help developers with information overload and situational awareness. Kumari and Singh's paper

appears to use "naïve Bayes (NB) and Deep Learning (DL) using entropy based measures for bug priority prediction" and show promising results that the classifiers trained on issues labelled with their priority and modified by their issue description entropy can predict an unknown bug's priority. This study does suffer from unclear experimental methods, very little rationalisation for design decisions, and only a single repository (OpenOffice) as input data.

The significance of our paper is that we present the first analysis of the consequences of surprisal in issue trackers. We describe how it relates to human perception of surprisal and how it may steer 'intuitive' prioritisation. We also combine the previous work in psycho-linguistics that shows decreased reading comprehension correlating to surprisal with the difficulty of resolving issues that contain surprising descriptions. Additionally, we look at how important surprising issues are, in an effort to determine if they should be foremost presented to developers looking for greater situational awareness of the development process. The results of this study suggest that when mining software repositories, the surprisal represented in issues is not useful in determining or assisting the development process.

10 Implications

In this section, we summarize the main motivation, contribution, and key findings of our work along with the implications that our work has for guiding future research.

Main Contribution: Motivated by the need for developers to stay informed about unexpected or unusual events in software projects [58], this work conceptualizes 'surprisal' in the context of software development and investigates whether it aligns with traditional information theory, specifically through experiments on issue tracking data.

Key Finding: The study finds that surprisal, as conceptualized, does not strongly align with key information-theoretic predictions, particularly in the context of difficulty and importance of issues in software development.

Implications: The results of this study suggest that surprisal is not an effective metric with which to rate the difficulty or importance of issues in issue trackers. While surprisal may still indicate anomalies, the consequences of these are not represented by the metrics we test against. Surprisal analysis may yet still provide a useful technique for issue tracker mining in three typical use cases, issue quality analysis, discussion analysis, and structure analysis [46], §3, but perhaps not with statistical language models.

Since the motivation for the work has not changed, future work should focus on refining how surprisal is calculated in an automated way. We outline several specific directions:

- Beyond Issue Titles and Descriptions: Future research should move beyond analyzing just issue titles and descriptions. This study shows that surprisal, when applied to these unstructured textual elements, does not strongly correlate with issue difficulty or importance. Future work could build on prior studies like Treude, Figueira Filho, and Kulesza's work [58], which focused on issue metrics such as the number of comments or labels, without considering the content of the issues themselves. Combining surprisal with these metrics may yield a more effective model for predicting issue prioritization and developer awareness.
- Incorporating Code and Other Structured Data: Since textual surprisal alone is insufficient, future work should explore surprisal metrics based on structured data such as code changes in pull requests, diffs, or commits. Examining how surprising code changes influence developer workflows or the overall health of a repository could extend this research.
- Inclusion of Visual and Multimodal Data: Issues frequently include images, diagrams, or log files, which were not considered in this study. Future models could integrate these forms of multimodal data to better predict developer surprise and issue difficulty, making surprisal metrics more accurate and reflective of real-world issue resolution processes.
- Context-Aware Surprisal Models: Developers rely heavily on context, such as prior issues, commit histories, and project timelines, when assessing issues. Future work could explore how surprisal metrics might improve by incorporating this historical context, allowing models to detect patterns or anomalies over time rather than treating each issue in isolation.
- Cross-Project and Longitudinal Studies: Expanding surprisal research to multiple repositories and over extended periods could reveal how surprisal evolves within and across projects. A cross-project analysis would help determine whether surprisal can be generalized across different types of software projects or if it is more applicable in certain domains.
- Surprisal and Developer Expertise: This study treated surprisal uniformly across different types of developers. Future research should consider how developer expertise or familiarity with a project affects their perception of surprising issues. Tailoring surprisal metrics based on developer experience could provide more actionable and context-sensitive insights.
- Surprisal and Developer Comprehension: Surprising issue descriptions can negatively affect comprehension, aligning with previous research in psycholinguistics [27, 42, 20]. Future work could explore how surprisal impacts the cognitive load of developers when processing issue descriptions, which could have implications for designing issue tracking systems and writing guidelines to improve clarity and ease of understanding.

11 Conclusion

In this study, we have explored the statistical relationships between the surprisal of issues and pull requests in software repositories and its correlation with

events a developer would want to be aware of. To measure such events, we have used two sets of proxies: resolution difficulty and perceived importance. Our resolution difficulty proxy combines reopening count, count of posts discussing the issue/PR, and time to resolution. Our perceived importance proxy combines mention in release notes, speed to address the issue/PR, number of GitHub reactions, and the assignment of high priority labels to the issue/PR.

We hypothesized that issues and pull requests with higher surprisal would be correlated with greater difficulty and higher perceived importance, under these proxies. However, our results did not support these hypotheses. Despite the hints of potential correlations suggested, we did not find significant correlations using our proxies. Better proxies and other events, like edits to documentation, requirements, or specification, will need to be explored to definitively confirm or confute this work's central hypothesis.

12 Conflicts of Interest

The authors declared that they have no conflicts of interest.

13 Data Availability Statement

The datasets generated by the researchers and the code used in the analysis are both available in the Zenodo repository[10], `https://doi.org/10.5281/zenodo.10647204`.

References

1. Abebe, S.L., Ali, N., Hassan, A.E.: An empirical study of software release notes. Empirical Software Engineering **21**(3), 1107–1142 (2016)
2. Agarwal, A., Gupta, N.: Comparison of outlier detection techniques for structured data (2021). URL `https://arxiv.org/abs/2106.08779`
3. Allamanis, M., Barr, E.T., Devanbu, P., Sutton, C.: A survey of machine learning for big code and naturalness. ACM Comput. Surv. **51**(4) (2018). DOI 10.1145/3212695. URL `https://doi.org/10.1145/3212695`
4. Baysal, O., Holmes, R., Godfrey, M.W.: Situational awareness: Personalizing issue tracking systems. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 1185–1188 (2013). DOI 10.1109/ICSE.2013.6606674
5. Baysal, O., Holmes, R., Godfrey, M.W.: No issue left behind: reducing information overload in issue tracking. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, p. 666–677. Association for Computing Machinery, New York, NY, USA (2014). DOI 10.1145/2635868.2635887. URL `https://doi.org/10.1145/2635868.2635887`
6. Bi, T., Xia, X., Lo, D., Grundy, J., Zimmermann, T.: An empirical study of release note production and usage in practice. IEEE Transactions on Software Engineering **48**(6), 1834–1852 (2022). DOI 10.1109/TSE.2020.3038881
7. Borges, H., Brito, R., Valente, M.T.: Beyond textual issues: Understanding the usage and impact of github reactions. In: Proceedings of the Brazilian Symposium on Software Engineering, pp. 397–406. ACM, New York, NY, United States (2019)

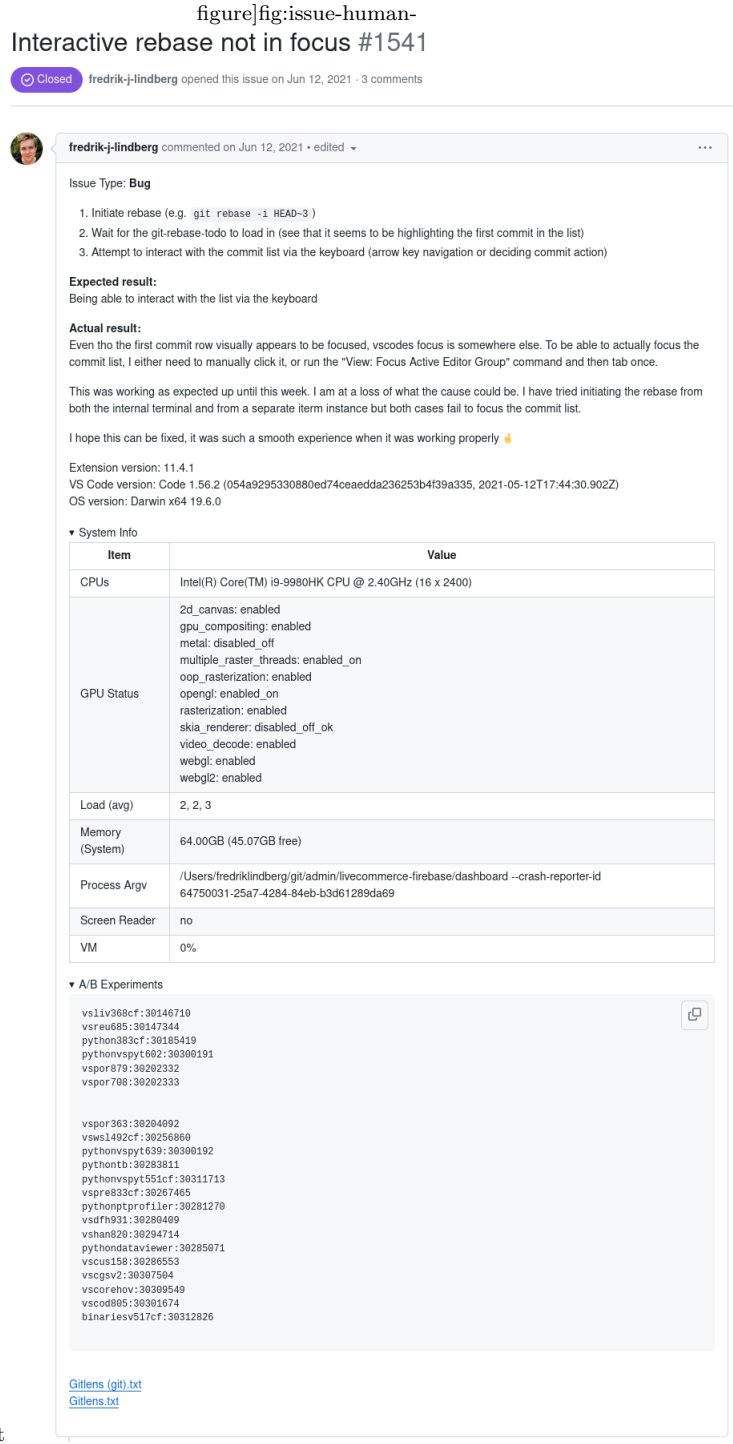
8. Borges, H., Hora, A., Valente, M.T.: Understanding the factors that impact the popularity of github repositories. In: Proceedings of the International Conference on Software Maintenance and Evolution, pp. 334–344. IEEE, Los Alamitos, CA, United States (2016)
9. Bugayenko, Y., Bakare, A., Cheverda, A., Farina, M., Kruglov, A., Plaksin, Y., Pedrycz, W., Succi, G.: Prioritizing tasks in software development: A systematic literature review. PLOS ONE **18**(4), 1–31 (2023). DOI 10.1371/journal.pone.0283838. URL https://doi.org/10.1371/journal.pone.0283838
10. Caddy, J.: The Role of Surprisal in Issue Trackers - Software and Datasets (2024). DOI 10.5281/zenodo.10647204. URL https://doi.org/10.5281/zenodo.10647204
11. Caddy, J., Treude, C.: Prioritising github priority labels. arXiv preprint arXiv:2405.10891 (2024)
12. Caddy, J., Wagner, M., Treude, C., Barr, E.T., Allamanis, M.: Is surprisal in issue trackers actionable? arXiv preprint arXiv:2204.07363 (2022)
13. Chen, S.F., Goodman, J.: An empirical study of smoothing techniques for language modeling. Computer Speech & Language **13**(4), 359–394 (1999)
14. Chyung, S.Y.Y., Roberts, K., Swanson, I., Hankinson, A.: Evidence-based survey design: The use of a midpoint on the likert scale. Performance Improvement **56**(10), 15–23 (2017). DOI https://doi.org/10.1002/pfi.21727. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/pfi.21727
15. D'Avila, L.F., Barbosa, J.L.V., de Oliveira, K.S.F.: Sw-context: a model to improve developers' situational awareness. IET Software **14**(5), 535–543 (2020). DOI https://doi.org/10.1049/iet-sen.2018.5156. URL https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2018.5156
16. De Boer, P.T., Kroese, D.P., Mannor, S., Rubinstein, R.Y.: A tutorial on the cross-entropy method. Annals of operations research **134**, 19–67 (2005)
17. Docs, G.: Issue event types (2022). URL https://docs.github.com/en/developers/webhooks-and-events/events/issue-event-types
18. Docs, G.: Rate limit (2023). URL https://docs.github.com/en/rest/issues/issues#get-an-issue
19. Fano, R.M.: The transmission of information. Massachusetts Institute of Technology, Research Laboratory of Electronics, Cambridge, MA, United States (1949)
20. Fernandez Monsalve, I., Frank, S.L., Vigliocco, G.: Lexical surprisal as a general predictor of reading time. In: W. Daelemans (ed.) Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, pp. 398–408. Association for Computational Linguistics, Avignon, France (2012). URL https://aclanthology.org/E12-1041
21. vscode-gitlens GitHub Repository: vscode-gitlens issue #1541 (2021). URL https://github.com/gitkraken/vscode-gitlens/issues/1541
22. vscode-gitlens GitHub Repository: vscode-gitlens pull request #2040 (2022). URL https://github.com/gitkraken/vscode-gitlens/pull/2040
23. Gonzalez, D., Zimmermann, T., Godefroid, P., Schäfer, M.: Anomalicious: Automated detection of anomalous and potentially malicious commits on github. In: Proceedings of the International Conference on Software Engineering: Software Engineering in Practice, pp. 258–267. IEEE, Los Alamitos, CA, United States (2021)
24. Goyal, R., Ferreira, G., Kästner, C., Herbsleb, J.: Identifying unusual commits on github. Journal of Software: Evolution and Process **30**(1), e1893 (2018)
25. Group, W.W.: Character model for the world wide web: String matching (2021). URL https://www.w3.org/TR/charmod-norm/#{}normalizationChoice
26. Hale, J.: A probabilistic earley parser as a psycholinguistic model. In: Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies, NAACL '01, p. 1–8. Association for Computational Linguistics, USA (2001). DOI 10.3115/1073336.1073357. URL https://doi.org/10.3115/1073336.1073357
27. Hale, J.: A probabilistic earley parser as a psycholinguistic model. In: Second meeting of the north american chapter of the association for computational linguistics (2001)
28. Hayes, A.F., Krippendorff, K.: Answering the call for a standard reliability measure for coding data. Communication Methods and Measures **1**(1), 77–89 (2007). DOI 10.1080/19312450709336664. URL https://doi.org/10.1080/19312450709336664

29. He, H., Peng, N., Liang, P.: Pun generation with surprise. In: J. Burstein, C. Doran, T. Solorio (eds.) Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 1734–1744. Association for Computational Linguistics, Minneapolis, Minnesota (2019). DOI 10.18653/v1/N19-1172. URL https://aclanthology.org/N19-1172
30. Hindle, A., Barr, E.T., Gabel, M., Su, Z., Devanbu, P.: On the naturalness of software. Communications of the ACM **59**(5), 122–131 (2016)
31. Izadi, M., Akbari, K., Heydarnoori, A.: Predicting the objective and priority of issue reports in software repositories. Empirical Software Engineering **27**(2), 50 (2022)
32. Jiang, J., Mohamed, A., Zhang, L.: What are the characteristics of reopened pull requests? a case study on open source projects in github. IEEE Access **7**, 102751–102761 (2019)
33. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The promises and perils of mining github. In: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), pp. 92–101. ACM, New York, NY, United States (2014)
34. Kao, J.T., Levy, R., Goodman, N.D.: A computational model of linguistic humor in puns. Cognitive Science **40**(5), 1270–1285 (2016). DOI https://doi.org/10.1111/cogs.12269. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.12269
35. Kavaler, D., Sirovica, S., Hellendoorn, V., Aranovich, R., Filkov, V.: Perceived language complexity in github issue discussions and their effect on issue resolution. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 72–83. IEEE, Los Alamitos, CA, United States (2017)
36. Kraemer, H.C., Kupfer, D.J., Clarke, D.E., Narrow, W.E., Regier, D.A.: Dsm-5: How reliable is reliable enough? American Journal of Psychiatry **169**(1), 13–15 (2012). DOI 10.1176/appi.ajp.2011.11010050. URL https://doi.org/10.1176/appi.ajp.2011.11010050. PMID: 22223009
37. Kullback, S.: Information theory and statistics. Wiley publication in mathematical statistics. Wiley, New York (1959)
38. Kumari, M., Singh, V.B.: An improved classifier based on entropy and deep learning for bug priority prediction. In: A. Abraham, A.K. Cherukuri, P. Melin, N. Gandhi (eds.) Intelligent Systems Design and Applications, pp. 571–580. Springer International Publishing, Cham (2020)
39. Kutner, M.H., Nachtsheim, C.J., Neter, J., et al.: Applied linear regression models, vol. 4. McGraw-Hill/Irwin, New York, NY, United States (2004)
40. Leite, L., Treude, C., Figueira Filho, F.: UEDashboard: Awareness of unusual events in commit histories. In: Proceedings of the Joint Meeting on Foundations of Software Engineering, pp. 978–981. ACM, New York, NY, United States (2015)
41. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of massive data sets. Cambridge university press (2020)
42. Levy, R.: Expectation-based syntactic comprehension. Cognition **106**(3), 1126–1177 (2008). DOI https://doi.org/10.1016/j.cognition.2007.05.006. URL https://www.sciencedirect.com/science/article/pii/S0010027707001436
43. Mikolov, T.: Statistical language models based on neural networks. Ph.d. thesis, Brno University of Technology, Faculty of Information Technology (2012). URL https://www.fit.vut.cz/study/phd-thesis/283/
44. Miller, R.C., Myers, B.A.: Outlier finding: focusing user attention on possible errors. In: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology, UIST '01, p. 81–90. Association for Computing Machinery, New York, NY, USA (2001). DOI 10.1145/502348.502361. URL https://doi.org/10.1145/502348.502361
45. Mohamed, A., Zhang, L., Jiang, J., Ktob, A.: Predicting which pull requests will get reopened in github. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp. 375–385. IEEE, Los Alamitos, CA, United States (2018)
46. Montgomery, L., Lüders, C., Maalej, W.: Mining issue trackers: Concepts and techniques (2024). URL https://arxiv.org/abs/2403.05716
47. Pang, Y., Xue, X., Namin, A.S.: Predicting vulnerable software components through n-gram analysis and statistical feature selection. In: 2015 IEEE 14th International

Conference on Machine Learning and Applications (ICMLA), pp. 543–548 (2015). DOI 10.1109/ICMLA.2015.99
48. Ray, B., Hellendoorn, V., Godhane, S., Tu, Z., Bacchelli, A., Devanbu, P.: On the "naturalness" of buggy code. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 428–439 (2016). DOI 10.1145/2884781.2884848
49. Romano, S., Caulo, M., Buompastore, M., Guerra, L., Mounsif, A., Telesca, M., Baldassarre, M.T., Scanniello, G.: G-repo: a tool to support msr studies on github. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 551–555. IEEE, Los Alamitos, CA, United States (2021). DOI 10.1109/SANER50967.2021.00064
50. Rosenfeld, R.: Two decades of statistical language modeling: where do we go from here? Proceedings of the IEEE **88**(8), 1270–1278 (2000). DOI 10.1109/5.880083
51. Seo, Y.S., Yoon, K.A., Bae, D.H.: An empirical analysis of software effort estimation with outlier elimination. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, PROMISE '08, p. 25–32. Association for Computing Machinery, New York, NY, USA (2008). DOI 10.1145/1370788.1370796. URL `https://doi.org/10.1145/1370788.1370796`
52. Shafiq, S., Mashkoor, A., Mayr-Dorn, C., Egyed, A.: Nlp4ip: Natural language processing-based recommendation approach for issues prioritization. In: 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 99–108 (2021). DOI 10.1109/SEAA53835.2021.00022
53. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal **27**(3), 379–423 (1948). DOI https://doi.org/10.1002/j.1538-7305.1948.tb01338.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x`
54. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika **52**(3/4), 591–611 (1965)
55. Singh, A., Kapur, P., Singh, V.: Developing classifiers by considering sentiment analysis of reported bugs for priority prediction. International Journal of System Assurance Engineering and Management pp. 1–12 (2023)
56. Spearman, C.: The proof and measurement of association between two things. The American Journal of Psychology **100**(3/4), 441–471 (1987)
57. Standard, H.L.: HTML Living Standard: Void Elements (2022). URL `https://html.spec.whatwg.org/#void-elements`
58. Treude, C., Figueira Filho, F., Kulesza, U.: Summarizing and measuring development activity. In: Proceedings of the Joint Meeting on Foundations of Software Engineering, pp. 625–636. ACM, New York, NY, United States (2015)
59. Umer, Q., Liu, H., Sultan, Y.: Emotion based automated priority prediction for bug reports. IEEE Access **6**, 35743–35752 (2018). DOI 10.1109/ACCESS.2018.2850910
60. Vignero, L., Demey, L.: The perfect surprise: a new analysis in dynamic epistemic logic. Logic Journal of the IGPL **28**(3), 341–362 (2019). DOI 10.1093/jigpal/jzz031. URL `https://doi.org/10.1093/jigpal/jzz031`
61. Wang, J., Zhang, X., Chen, L.: How well do pre-trained contextual language representations recommend labels for github issues? Knowledge-Based Systems **232**, 107476 (2021)
62. Wang, S., Chollak, D., Movshovitz-Attias, D., Tan, L.: Bugram: bug detection with n-gram language models. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE '16, p. 708–719. Association for Computing Machinery, New York, NY, USA (2016). DOI 10.1145/2970276.2970341. URL `https://doi.org/10.1145/2970276.2970341`
63. Wessel, J.R., Danielmeier, C., Morton, J.B., Ullsperger, M.: Surprise and error: Common neuronal architecture for the processing of errors and novelty. Journal of Neuroscience **32**(22), 7528–7537 (2012). DOI 10.1523/JNEUROSCI.6352-11.2012. URL `https://www.jneurosci.org/content/32/22/7528`
64. Xie, X., Su, Y., Chen, S., Chen, L., Xuan, J., Xu, B.: Mula: A just-in-time multi-labeling system for issue reports. IEEE Transactions on Reliability **1**(1), 1–14 (2021)
65. Yazici, B., Asma, S.: A comparison of various tests of normality. Journal of Statistical Computation and Simulation - J STAT COMPUT SIM **77**, 175–183 (2007). DOI 10.1080/10629360600678310

Fig. 11 An example of a Github issue from the manual rating set that showed a large amount of disagreement between the human raters. [21]

figure]fig:issue-human-



Interactive rebase not in focus #1541

Closed **fredrik-j-lindberg** opened this issue on Jun 12, 2021 · 3 comments

fredrik-j-lindberg commented on Jun 12, 2021 • edited

Issue Type: **Bug**

1. Initiate rebase (e.g. `git rebase -i HEAD~3`)
2. Wait for the git-rebase-todo to load in (see that it seems to be highlighting the first commit in the list)
3. Attempt to interact with the commit list via the keyboard (arrow key navigation or deciding commit action)

Expected result:
Being able to interact with the list via the keyboard

Actual result:
Even tho the first commit row visually appears to be focused, vscodes focus is somewhere else. To be able to actually focus the commit list, I either need to manually click it, or run the "View: Focus Active Editor Group" command and then tab once.

This was working as expected up until this week. I am at a loss of what the cause could be. I have tried initiating the rebase from both the internal terminal and from a separate iterm instance but both cases fail to focus the commit list.

I hope this can be fixed, it was such a smooth experience when it was working properly ✌️

Extension version: 11.4.1
VS Code version: Code 1.56.2 (054a9295330880ed74ceaedda236253b4f39a335, 2021-05-12T17:44:30.902Z)
OS version: Darwin x64 19.6.0

▾ System Info

Item	Value
CPUs	Intel(R) Core(TM) i9-9980HK CPU @ 2.40GHz (16 x 2400)
GPU Status	2d_canvas: enabled gpu_compositing: enabled metal: disabled_off multiple_raster_threads: enabled_on oop_rasterization: enabled opengl: enabled_on rasterization: enabled skia_renderer: disabled_off_ok video_decode: enabled webgl: enabled webgl2: enabled
Load (avg)	2, 2, 3
Memory (System)	64.00GB (45.07GB free)
Process Argv	/Users/fredriklindberg/git/admin/livecommerce-firebase/dashboard --crash-reporter-id 64750031-25a7-4284-84eb-b3d61289da69
Screen Reader	no
VM	0%

▾ A/B Experiments

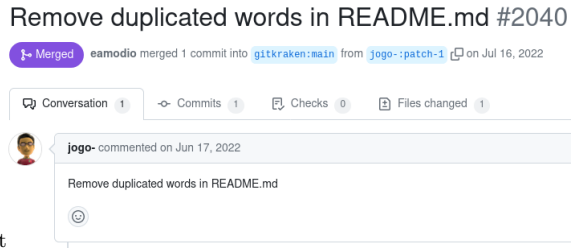
```
vsliv368cf:30146710
vsreu685:30147344
python383cf:30185419
pythonvspyt602:30300191
vspor879:30202332
vspor708:30202333


vspor363:30204092
vswsl492cf:30256860
pythonvspyt639:30300192
pythontb:30283811
pythonvspyt551cf:30311713
vspre833cf:30267465
pythonptprofiler:30281270
vsdfh931:30280409
vshan820:30294714
pythondataviewer:30285071
vscus158:30286553
vscgsv2:30307504
vscorehov:30309549
vscod805:30301674
binariesv517cf:30312826
```

Gitlens (git).txt
Gitlens.txt

disagreement

Fig. 12 An example of a Github issue where the machine rating of high surprisal is from unusual, but insignificant words. [22]

figure]fig:issue-machine-



disagreement