Is Surprisal in Issue Trackers Actionable?

James Caddy
University of Adelaide
Adelaide, Australia
james.caddy@adelaide.edu.au

Markus Wagner
University of Adelaide
Adelaide, Australia
markus.wagner@adelaide.edu.au

Christoph Treude
University of Melbourne
Melbourne, Australia
christoph.treude@unimelb.edu.au

Earl T. Barr
University College London
London, United Kingdom
e.barr@ucl.ac.uk

Miltiadis Allamanis
Microsoft Research
Cambridge, United Kingdom
miltos@allamanis.com

ABSTRACT

Background. From information theory, surprisal is a measurement of how unexpected an event is. Statistical language models provide a probabilistic approximation of natural languages, and because surprisal is constructed with the probability of an event occuring, it is therefore possible to determine the surprisal associated with English sentences. The issues and pull requests of software repository issue trackers give insight into the development process and likely contain the surprising events of this process.

Objective. Prior works have identified that unusual events in software repositories are of interest to developers, and use simple code metrics-based methods for detecting them. In this study we will propose a new method for unusual event detection in software repositories using surprisal. With the ability to find surprising issues and pull requests, we intend to further analyse them to determine if they actually hold importance in a repository, or if they pose a significant challenge to address. If it is possible to find bad surprises early, or before they cause additional troubles, it is plausible that effort, cost and time will be saved as a result.

Method. After extracting the issues and pull requests from 5000 of the most popular software repositories on GitHub, we will train a language model to represent these issues. We will measure their perceived importance in the repository, measure their resolution difficulty using several analogues, measure the surprisal of each, and finally generate inferential statistics to describe any correlations.

CCS CONCEPTS

• **Information systems** → **Language models**; *Data mining*; • **Computing methodologies** → *Anomaly detection*; Maximum likelihood modeling.

KEYWORDS

self-information, n-gram, GitHub issues

1 INTRODUCTION

Surprisal is a measure in information theory that can quantify how unexpected and thus how informative a word w_t is given the words that precede it ($w_1, ..., w_{t-1}$). A higher word surprisal value indicates that the current word is less expected given the context. In mathematical terms, surprisal is defined as the negative logarithm of the word's conditional probability of occurrence [18].

In this work, we propose to apply the surprisal measure to software engineering artefacts, motivated by many researchers arguing that software developers need to be aware of unusual or surprising events in their repositories, e.g., when summarizing project activity [19], notifying developers about unusual commits [7, 9], and for the identification of malicious content [26]. The basic intuition is that catching bad surprises early will save effort, cost, and time, since bugs cost significantly more to fix during implementation or testing than in earlier phases [17], and by extension, bugs cost more the longer they exist in a product after being reported and before being addressed.

Following recent work on applying natural language techniques to software engineering data [14], in this work, we investigate whether the information-theoretic measure of surprisal is actionable when applied to software repositories. In this study we analyse issues and pull requests separately, but as a matter of convention, we will refer to these simply as issues going forward. In our method, the differences are negligible, but separately they may produce unique analyses.

We investigate two scenarios involving surprisal: its effect on resolution difficulty and the perceived importance of surprising issues. Prior work [23] found that conformance to project-specific language norms reduces issue resolution time. Assuming that surprising issues do not conform to such norms, we investigate whether they are more difficult to resolve. Other work [22] has analysed what kind of issues are reopened, and specific issue metrics such as number of comments are shown to correlate. We conceptualise difficulty in terms of (1) reopened rate, (2) amount of discussion, and

(3) resolution time. Further, prior work established that developers want to be aware of unexpected issues in their repositories [19], and that high importance issues are more likely to be included in release notes [12]. In this work, we investigate whether surprising issues are treated with higher importance. We conceptualise importance in terms of (1) mention in release notes, (2) first issues to be worked on after a break, (3) issues attracting GitHub reactions [15], and (4) priority labels being assigned to issues.

1.1 Motivating Examples

Take for example, a software product sustainment team that receives a steady stream of issue reports which they must triage. Each of these issues have a cost increasing at a given rate, which itself might be accelerating. This cost may be in safety, budget, and/or reputation for example. It is in the interest of stakeholders that cost is minimised across the lifetime of the product.

In order to predict the cost of the new issues coming in, someone with experience needs to spend time comparing against previous experience, or applying metrics based methods. Then time needs to be spent addressing the issue for the cost to be eliminated.

If surprisal can be used to tell which issues are challenging or notable without human input, it is possible that efficiency can be gained in the resolution process. The triage process is better informed with notable issues that might need to be prioritised sooner. The cost to address an issue, weighed against its cost to the project, is better informed by the challenge that the issue presents. Challenging issues can have more people assigned to resolve it. Information about notable issues can be distributed to other developers so they know what mistakes to avoid in the future.

Another example is of a new developer just joining a project. In order to familiarise themselves with the history and most important changes or developments that have occurred, they would usually have to rely on release notes or a version history. In the case where the project either lacks release notes, or in the case where the release notes contain even the most minor changes, this can be overwhelming and of limited value to the new developer. A tool based on surprisal could extract the most important or notable changes from a mature project for this developer, or even tell someone who has been away from a project for a period what notable things have happened since they left.

2 BACKGROUND

In Claude Shannon's seminal work on information theory he describes radio signals and their ability to communicate information. In doing so, he makes the first formal description of information entropy or information uncertainty.

2.1 Uncertainty of an Event

Take two events, represented by the symbols 'A' and 'B'. If we know that these events are equally likely to occur, we could say that we are equally uncertain which the next event will be. The information that we gain from observing an event can be represented by $I(x) = -\log_2 P(x)$, where $P(x)$ is the probability of x occurring. In this example, we can see that for either outcome, $P(A) = P(B) = 0.5$ and so $I(A) = I(B) = 1$. We gain exactly 1 bit

of information, as there are two possible outcomes and either is as likely to occur.

If we take another example, and modify the probabilities of the outcomes such that $P(A) = 0.0$ and $P(B) = 1.0$, we can see that if we observe B, we gain exactly 0 bits of information from $I(B) = -\log_2 1 = 0$. This is intuitive, since we already knew that the event would be B, there was no uncertainty.

Now we take an example where an event is extremely unlikely to happen. When $P(A) = 0.999$ and $P(B) = 0.001$, if we observe the event A, we gain $I(A) = -\log_2 0.999 \approx 0.001$ bits of information. Since it was likely to happen, we do not receive much information, but still a little. It is then surprising when we observe the event B, since it is unlikely to happen. The information we gain from observing B, $I(B) = -log_2 0.001 \approx 9.966$, is extreme in comparison.

2.2 Statistical Language Models

Shannon describes an approximation of the English language by utilising n-gram statistical language models. Initially he describes a crude unigram model that selects the next word in a sentence based on their relative frequencies in the English language. This has an obvious flaw; unigram models assume that each choice of word is independent from the last. Syntax and grammar demand more careful choice of words than pure randomness alone, so to account for this, greater order n-gram models are used.

Bigram models select the next word based on the previous word, and trigrams select based on the previous two. These better capture the structure of English, and while higher order models are possible, these tend to overfit the training data. The language model also becomes exponentially sparser as the order increases, which will in turn require exponentially more training data or risk severely underfitting the training data.

With a model that statistically represents the English language in n-grams, we can now determine how likely a word w_t is, given the words that precede it ($w_1, ..., w_{t-1}$).

2.3 Probability Distributions

To determine how closely an issue's description represents the whole corpus of descriptions, in an effort to see how surprising or not it is, we can use cross entropy. To understand cross entropy, it is important to discuss the underlying concept of entropy. Entropy can be quantified as the average number of symbols needed to represent an event from a distribution of events. Take for example, a distribution of events where:

$$P(A) = 0.5,$$
$$P(B) = 0.25,$$
$$P(C) = 0.25$$

This distribution is biased, or skewed in favour of observing the event 'A'. We can describe these events with their relative frequencies in a Shannon–Fano coding [5], where:

$$A = \{0\},$$
$$B = \{01\},$$
$$C = \{10\}$$

In this example, we can see that half of the time, we only require 1 bit of information to represent any event. The other half of the time, we require 2 bits. We can therefore say that the average number of bits required to represent an event from this distribution (its entropy) is 1.5. Just as above, where an event that occurs often can be represented with less bits, distributions that are more biased, that yield one particular event more often, present less entropy. Shannon formalises this into the following equation, using X as the support for random events x:

$$H(P) = - \sum_{x \in X} P(x) \times \log(P(x)) \qquad (1)$$

Cross entropy describes 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.

$$H(P_o, P_{tt}) = - \sum_{x \in X} P_o(x) \times \log(P_{tt}(x)) \qquad (2)$$

When $P_o = P_{tt}$, Equation (2) is equivalent to Equation (1). Otherwise, 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.

There are potentially other surprisal metrics that provide different evaluations. More simple measures for instance, might take the minimum, maximum, or average surprisal of all words in an issue. Cross entropy is a very well-realised metric of model accuracy in the literature [11], and it is for this reason that we are using it.

3 RESEARCH QUESTIONS

A number of research questions and hypotheses have been made to guide the investigation.

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, and also which factors of the language model influence its ability to measure the surprisal of an issue.

RQ2: Are surprising issues correlated with resolution difficulty?

In RQ2, we define "resolution difficulty" as a combination of the following factors. Difficult to resolve issues are: reopened more often; attract more discussion prior to resolution; and take longer to be resolved, when compared to issues with little or no resolution difficulty. This is perhaps an incomplete list, but will serve as the basis for the report.

RQ2 can be formalised 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 an unsurprising issue.

$H_{2.4}$: 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: Are surprising pull requests correlated with resolution difficulty?

As in RQ2, RQ3 defines difficulty in the same way but for pull requests rather than issues. The purpose of RQ3 is to determine if the more structured and formal contents of pull requests are more suitable than issues for establishing a relationship between surprisal and difficulty. Pull requests offer a suitable alternative because they are intended to directly address a need that would typically be expressed in an issue. Additionally, merged pull requests are reviewed and thus are unlikely to be duplicated, describe what they are resolving, and what they address is far less likely to be misreported as a defect if it is not.

The formal hypotheses for RQ3 are formulated in a manner identical to the hypotheses of RQ2, since pull requests and issues are functionally identical in this context. In the interest of brevity, the full text of these hypotheses $H_{2.1}$ through $H_{2.5}$ has been omitted.

RQ4: Are surprising issues correlated with perceived importance?

In RQ4, we define "perceived importance" as a combination of the following factors. Important issues are: mentioned in release notes; addressed soon after periods of breaks; attract more GitHub reactions; and have high-priority labels added. Once again, this is perhaps an incomplete list, but will serve as the basis for the report.

RQ4 can be formalised 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$_{3.2}$, we define a hiatus as the top 25% longest times between issue resolutions per contributor for a particular repository. Priority in this case is the order in which issues are worked on after a hiatus.

> **RQ5**: Are surprising pull requests correlated with perceived importance?

As before with RQ2 and RQ3, RQ4 focuses on issues, and RQ5 will focus on their pull request counterparts. The full text of hypotheses H$_{4.1}$ through H$_{4.4}$ has been omitted for brevity.

4 VARIABLES

A summary of the variables involved for all the research questions can be found in Table 1.

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.

Response Variables:

- **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 [8], 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.* [23] 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 [3]. 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.* [23]

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; 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 [12].
- **Order of Address.** After lengthy breaks of development, whether independently taken or due to holidays, it is likely that the most pressing of issues in the backlog are chosen for immediate resolution. Commits after extended breaks have been described as interesting, in a previous paper [19]. Each contributor has a time between addressing issues, so taking the top 25% of these breaks, and then ordering the issues that they worked on afterwards gives us an indication of what importance that author places on each issue.
- **Reactions.** How many reactions have been made on the issue. Reactions give a quick way for users 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. This can be seen as a community rating of importance, rather than that of the maintainers [15].
- **Labelling.** Many repositories use the GitHub issue labelling system to triage incoming bug reports and feature requests. Issues are sorted by maintainers into priorities and labelled as such, from low priority to high priority [1], §3.B. These labels can be seen as the maintainer's rating of importance.

5 DATA SETS

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

5.1 Sources

For this study, open-source software stored on GitHub serves as our primary and sole source for software issues. Unfortunately, GitHub imposes a restrictive limit to how many interactions with its API a user can make. Typically this is 5000 calls per hour [2], and so we take the top 5000 most 'starred' repositories as our data set. Stars represent a user liking a repository, and therefore indicate popular repositories, more likely to have high numbers of issues due to increased testing and feature requests — a result of more users [13].

This supposes that GitHub is used in the same way by the maintainers of those 5000 repositories, which is not the case. Many of these repositories do not make use of the Issues feature; many of these repositories are not software development related and possess little to no code; and other repositories are simply a mirror of a repository developed and hosted elsewhere [29]. Additionally, we

Table 1: Variables

Variable	Hypotheses	Description	Measure	Operationalisation
Surprisal	Predictor for all hypotheses	How surprising an issue is to the statistical language model.	Ratio	Cross entropy of issue, obtained with probability from SLM trained on corpus of all issues.
Reopenings	Response for $H_{1.1}$, $H_{2.1}$	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.	Ratio	GitHub Issue API's "reopened" event.
Participants	Response for $H_{1.2}$, $H_{2.2}$	Number of unique individuals involved with the issue. Each event (as described by the GitHub API [3]) associated with an issue has an actor that initiates it, whether that event is a comment, or a state update.	Ratio	GitHub Issue API's event "actor" for each event associated with an issue.
Interactions	Response for $H_{1.3}$, $H_{2.3}$	Number of events associated with the issue.	Ratio	Count of events, as described by the GitHub Issue API [3].
Open State Duration	Response for $H_{1.4}$, $H_{2.4}$	Length of time between the issue's creation, and it being resolved for the final time.	Ratio	Difference between GitHub Issue API's "created_at" value for the issue and final "closed" or "merged" event.
Mentions	Response for $H_{3.1}$, $H_{4.1}$	Number of mentions within a repository's release notes.	Ratio	Scrape for issue numbers through GitHub's Releases API.
Order of Address	Response for $H_{3.2}$, $H_{4.2}$	Issue order after top 25% of contributor's inter-issue resolution times.	Interval	Issues assigned to a contributor are retrieved through the GitHub Issues API.
Reactions	Response for $H_{3.3}$, $H_{4.3}$	Number of reactions on a particular issue.	Ratio	Count of reactions for an issue, from GitHub's Reactions API.
Labelling	Response for $H_{3.4}$, $H_{4.4}$	Assigned priority or importance label of a particular issue.	Interval	Labels are extracted via the GitHub Issues API, and then normalised (per repository) to a 3-degree scale, 'low-importance', 'regular-importance', 'high-importance'.

wish to limit the scope to English language repositories. To find the repositories that fit these conditions, we plan to use G-Repo [16] as a means to filter out non-software repositories, repositories making little use of the Issues feature (less than 1000 issues), and non-English repositories.

5.2 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 would see some improvement in accurately modelling the software language used, compared to a more general pre-trained model [4], §5.3. For this reason, it was decided that a bespoke language model will be trained for the task.

5.3 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 will prepend the issue description with the issue title. The SLM will then be 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 [27] 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 [10].
(4) Punctuation and symbols are removed on word boundaries, and when isolated.
(5) Stop words are removed, and remaining words are stemmed.
   (See details following.)

As singular code tokens — variable names and the like — may provide important contextual information across multiple issues, they will be preserved in the training data. 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 is 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).

It is possible that the quality of the language model could be improved by further transforming the text in a final step. In many natural language processing applications, stop word removal is conducted to remove low-information tokens, thereby increasing the average entropy. In other applications, lemmatisation or stemming is used to reduce the occurrence of high-information tokens. Concepts are learned better if you can reduce "fishing" and "fishlike" to "fish". However we cannot know for certain what using either will achieve when modelling surprisal, and a preliminary investigation will be conducted to see if it indeed increases the accuracy of the model. The choice of algorithms is still to be determined, and will be revisited in the final report.

5.3.1 Priority Labelling. 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'.

In order to process these priorities, it is necessary for manual classification to normalise these different ranges. For the purpose of this experiment, priority is distributed among the three degrees 'low-importance', 'regular-importance', and 'high-importance'. Tie-breaking is settled by ruling in favour of a higher importance, using the previous examples for example; 'low-importance' would take 'P1' and 'Low Priority'; 'regular-importance' would take 'P2' and 'P3'; and 'high-importance' would take 'P4', 'P5', and 'High Priority'.

To ensure that these have been correctly classified, we propose giving two researchers a list of 400 randomly selected labels from the population of approximately 9000 different labels and asking them to classify them using the method above, or classify them as 'unrelated' to priority or importance. The agreement between the researchers is then calculated using Cohen's kappa, and in the event that a consensus value of 0.7 is reached, we consider the task sufficiently unambiguous, and a single researcher can then be trusted to accurately represent the rest of the $\sim$ 9000 labels. If not, a more involved process needing agreement by more researchers is required to classify the labels.

6 EXECUTION PLAN

In this section we present the method we plan to use for the study. It is important that all the decisions made during the experiment will be recorded, along with intermediate outputs. For example: a record will be made listing the 5000 most starred repositories on GitHub at the time of the experiment; after repositories are removed when failing to meet selection criteria, those will also be

made into a record; and decisions such as how the total issue count was calculated will be made into a record too. All the code used will be made available in a GitHub repository for the final report.

6.1 Method

To test all hypotheses the following method is proposed:

(1) Using the GitHub API, query the 5000 most starred repositories on GitHub.
(2) Query the number of issues. If any repository has less than 1000 issues, remove it from the pool.
(3) Extract all GitHub issues from each repository.
(4) Train SLM on entire issue description corpus.
(5) For each repository:
  (a) Determine if it uses priority/importance labels, and normalise those labels into three degrees of importance. Ties broken in favour of being more important.
  (b) For each issue:
    (i) Determine and record surprisal of issue.
    (ii) Record how many re-openings have occurred.
    (iii) Record how many unique participants have interacted with the issue.
    (iv) Record how many interactions have been made with the issue.
    (v) Calculate and record open duration for the issue.
    (vi) Record number of reactions to the issue.
    (vii) Record normalised importance label, if any.
  (c) Parse all release notes for the repository, incrementing the mentions of a particular issue when it appears.
  (d) For each contributor:
    (i) Determine top 25% of inter-issue resolution times.
    (ii) Split all issues assigned to the contributor into bands following a break of at least the 25th percentile's length.
    (iii) Order the issues by oldest assigned time.
    (iv) Record the ordinal position of each issue in its inter-break band.
(6) Split issues from GitHub into pull requests and simple issues.
(7) Generate descriptive statistics for each repository according to analysis plan.
(8) Generate inferential statistics across entire issue data set according to analysis plan.

6.2 Statistical Language Modelling

N-gram models saw some popularity during the 2000s and 2010s, but gradually saw declining popularity due to their perceived contextual fragility [28](2000), and competitive neural language models being developed [21](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 [6]. More modern statistical approaches typically use a n-gram model utilising a backoff strategy to deal with unknown tokens.

We train the SLM using the entire corpus of issues. This gives us the advantage of never coming across an unknown token, as all the tokens that we generate surprisal values for are contained in
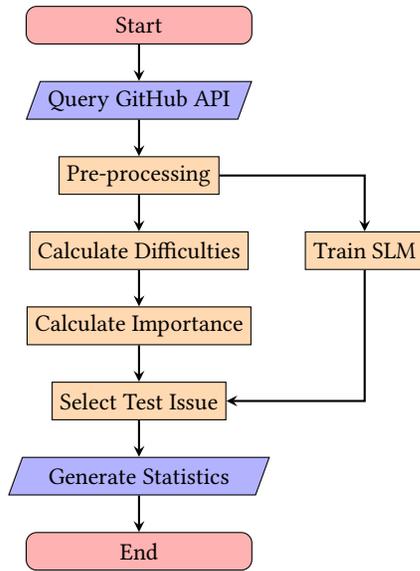
Figure 1: Method Flowchart

the training set. Any unknown, in this case 'unique', tokens would otherwise be infinitely surprising, which might not accurately represent how common it is to appear in say a larger sample size of repositories and issues.

The loss of generality realised by the lack of testing set is insignificant when compared to the small sample size of 5000. For this study, we are more concerned with the formulation of surprisal, and the two correlations applying to the repositories we have selected. Transferability can be further investigated after this proof of concept, if successful. Conversely, because singular repositories have a small sample size of issues, the specificity gained by training only on a single repository introduces massive sparsity in the model. This is another reason we include all issues into the training set.

6.3 Model Smoothing

Despite the lack of unknown tokens, we still apply a smoothing algorithm to obtain a more uniform, less sparse, and more representative model of the larger corpus of issues outside of the repositories that we have selected. The modified Kneser-Ney algorithm described by Chen and Goodman [6], §3, was chosen due to its excellent smoothing performance.

6.4 Model Improvement

While trigrams were chosen for the reasons previously mentioned, it is worth understanding how n-gram order affects the language model's calculation of surprisal. It is also worth understanding how the use of a testing set from the training data affects this calculation of surprisal.

These two factors contribute to the SLM and reveal important information regarding RQ1. In order to measure the affects of these factors, we propose the following additional experiment which compares the SLM performance against a manual classification:

(1) A repository of relatively few issues is chosen as a model.
(2) Take a representative sample of issues from the repository.
(3) Two researchers are asked to rate the surprisal in each issue, after reading all issues thoroughly.
  (a) Issues are rated on a Likert scale.
  (b) A value of 1 means that the description of an issue is not surprising.
  (c) A value of 5 means that the issue contains almost completely unique information.
  (d) Surprisal judgements should be based on factors such as topic, formatting, length, and word usage.
(4) Choose the SLM training set:
  (a) Entire issue descriptions corpus.
  (b) Entire issue descriptions corpus, with the chosen repository absent.
  (c) Entire issue descriptions corpus, with each issue removed individually.
(5) Choose n-gram order, repeating from 1-grams to 10-grams.
(6) Measure and record surprisal of each issue (the same individually removed from the training set if that type of training set is chosen).
(7) Generate agreement statistics between each model and the manual classification.

7 ANALYSIS PLAN

This is a correlational study of independent random variables, the following is a design of the quantitative analysis that will be undertaken. All calculations will be performed using a statistics package.

Model Analysis. In Section 6.4 we propose a method with which the surprisal of a repository-representative sample of issues is judged manually by two researchers. The surprisal of the issues is then measured using the surprisal calculated by the SLM. We propose using Cohen's kappa to measure the agreement between the two researchers, and report how contentious this task is. We also propose using a Kendall rank correlation to measure the agreement between the SLM- and manually-computed surprisal values. In the event that the agreement is statistically significant, it can be said that SLMs can calculate surprisal to a similar degree as a human participant can, and are suitable for the following analyses. A qualitative analysis of the issues that have disagreement in the surprisal ratings is conducted. Both the case of inter-researcher disagreement and researcher-SLM disagreement will be analysed in an effort to understand what caused this disagreement.

We expect that a language model using the entire corpus of issue descriptions and trigrams sufficiently agrees with the researcher's judgements. However, we will run the deeper analysis using different training sets and n-gram orders to confirm those choices are indeed correct. A combination of each choice will maximise the agreement with the manual computation, and this combination will be used in the analyses going forward. In the event that even the combination that maximises the agreement is still not statistically significant, a more thorough inspection of the factors influencing the SLM is required.

Descriptive Statistics. We will 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 will be included

Inferential Statistics. Before presenting a correlation and regression analysis using a linear regression, we graph the relationship between the surprisal and each response variable for the issues in a repository. We will then test each hypothesis separately with its corresponding variable, and look for statistical significance.

A Shapiro-Wilk test is used to determine if the data shows normality [24]. 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 [25], §1.2.a).

In order to test $H_{1.5}$ and $H_{2.5}$, we will 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 [20], §10.5, as our belief is that they represent the same variable: difficulty.

Interpretation of Results. We will present our interpretation of our findings, a discussion on any assumptions discovered, limitations, threats to validity, and future research.

REFERENCES

[1] Xiaoyuan Xie, Yuhui Su, Songqiang Chen, Lin Chen, Jifeng Xuan, and Baowen Xu. 2021. MULA: A Just-In-Time Multi-labeling System for Issue Reports. *IEEE Transactions on Reliability* 1, 1 (2021), 1–14.
[2] GitHub Docs. 2022. Rate limit. https://docs.github.com/en/rest/reference/rate-limit
[3] GitHub Docs. 2022. Issue event types. https://docs.github.com/en/developers/webhooks-and-events/events/issue-event-types
[4] Jun Wang, Xiaofang Zhang, and Lin Chen. 2021. How well do pre-trained contextual language representations recommend labels for GitHub issues? *Knowledge-Based Systems* 232 (2021), 107476.
[5] Robert M Fano. 1949. *The transmission of information*. Massachusetts Institute of Technology, Research Laboratory of Electronics, Cambridge, MA, United States.
[6] Stanley F Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* 13, 4 (1999), 359–394.
[7] Larissa Leite, Christoph Treude, and Fernando Figueira Filho. 2015. UEDashboard: Awareness of unusual events in commit histories. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*. ACM, New York, NY, United States, 978–981.
[8] Jing Jiang, Abdillah Mohamed, and Li Zhang. 2019. What are the characteristics of reopened pull requests? a case study on open source projects in github. *IEEE Access* 7 (2019), 102751–102761.
[9] Raman Goyal, Gabriel Ferreira, Christian Kästner, and James Herbsleb. 2018. Identifying unusual commits on GitHub. *Journal of Software: Evolution and Process* 30, 1 (2018), e1893.
[10] W3C Working Group. 2021. Character Model for the World Wide Web: String Matching. https://www.w3.org/TR/charmod-norm/#normalizationChoice
[11] Robert K. Niven. 2007. Combinatorial Information Theory: I. Philosophical Basis of Cross-Entropy and Entropy. arXiv:cond-mat/0512017 [cond-mat.stat-mech]
[12] Surafel Lemma Abebe, Nasir Ali, and Ahmed E Hassan. 2016. An empirical study of software release notes. *Empirical Software Engineering* 21, 3 (2016), 1107–1142.
[13] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the factors that impact the popularity of GitHub repositories. In *Proceedings of the International Conference on Software Maintenance and Evolution*. IEEE, Los Alamitos, CA, United States, 334–344.
[14] Abram Hindle, Earl T Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu. 2016. On the naturalness of software. *Commun. ACM* 59, 5 (2016), 122–131.
[15] Hudson Borges, Rodrigo Brito, and Marco Tulio Valente. 2019. Beyond Textual Issues: Understanding the Usage and Impact of GitHub Reactions. In *Proceedings of the Brazilian Symposium on Software Engineering*. ACM, New York, NY, United States, 397–406.
[16] Simone Romano, Maria Caulo, Matteo Buompastore, Leonardo Guerra, Anas Mounsif, Michele Telesca, Maria Teresa Baldassarre, and Giuseppe Scanniello. 2021. G-Repo: a Tool to Support MSR Studies on GitHub. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Los Alamitos, CA, United States, 551–555. https://doi.org/10.1109/SANER50967.2021.00064
[17] Maurice Dawson, Darrell Norman Burrell, Emad Rahim, and Stephen Brewster. 2010. Integrating software assurance into the software development life cycle (SDLC). *Journal of Information Systems Technology and Planning* 3, 6 (2010), 49–53.
[18] Kristijan Armeni, Roel M Willems, and Stefan L Frank. 2017. Probabilistic language models in cognitive neuroscience: Promises and pitfalls. *Neuroscience & Biobehavioral Reviews* 83 (2017), 579–588.
[19] Christoph Treude, Fernando Figueira Filho, and Uirá Kulesza. 2015. Summarizing and measuring development activity. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*. ACM, New York, NY, United States, 625–636.
[20] Michael H Kutner, Christopher J Nachtsheim, John Neter, et al. 2004. *Applied linear regression models*. Vol. 4. McGraw-Hill/Irwin, New York, NY, United States.
[21] Tomáš Mikolov. 2012. *Statistical Language Models Based on Neural Networks*. Ph.D. thesis. Brno University of Technology, Faculty of Information Technology. https://www.fit.vut.cz/study/phd-thesis/283/
[22] Abdillah Mohamed, Li Zhang, Jing Jiang, and Ahmed Ktob. 2018. Predicting which pull requests will get reopened in GitHub. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, Los Alamitos, CA, United States, 375–385.
[23] David Kavaler, Sasha Sirovica, Vincent Hellendoorn, Raul Aranovich, and Vladimir Filkov. 2017. 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)*. IEEE, Los Alamitos, CA, United States, 72–83.
[24] Samuel Sanford Shapiro and Martin B Wilk. 1965. An analysis of variance test for normality (complete samples). *Biometrika* 52, 3/4 (1965), 591–611.
[25] Charles Spearman. 1987. The proof and measurement of association between two things. *The American Journal of Psychology* 100, 3/4 (1987), 441–471.
[26] Danielle Gonzalez, Thomas Zimmermann, Patrice Godefroid, and Max Schäfer. 2021. Anomalicious: Automated Detection of Anomalous and Potentially Malicious Commits on GitHub. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*. IEEE, Los Alamitos, CA, United States, 258–267.
[27] HTML Living Standard. 2022. HTML Living Standard: Void Elements. https://html.spec.whatwg.org/#void-elements
[28] R. Rosenfeld. 2000. Two decades of statistical language modeling: where do we go from here? *Proc. IEEE* 88, 8 (2000), 1270–1278. https://doi.org/10.1109/5.880083
[29] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. ACM, New York, NY, United States, 92–101.