Assessing Software Understandability in Systems by Leveraging Fuzzy Method and Linguistic Analysis

Michael Shoga

University of Southern California

CSER 2019
conference on systems engineering research

Assessing Software Understandability in Systems by Leveraging Fuzzy Method and Linguistic Analysis

Celia Chen, **Michael Shoga**, Brian Li, Barry Boehm

Software Understandability

Definitions

- Developers: Complexity and readability of the source code
- ISO 9126: 2001, 6.3.1. defines software understandability as “the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.”
- Boehm: “a characteristic of software quality which means ease of understanding software systems.”
    - Understandability is a factor of software maintenance

Software Maintainability and Understandability

75-90% of business and command&control software and 50-80% of cyber-physical system software costs are incurred during maintenance.

Software maintenance is an essential part of a software life cycle

Highly understandable software produces robust and maintainable systems.

Problem with Measurement

A recent empirical study considered 121 new and existing metrics and found they had little to no correlation with code understandability.

Some of the considered metrics included

- LOC
- Cyclomatic complexity
- Text coherence

Is there an alternative to the source code based approaches?

Bugs

Bugs are used widely in open source software as an indication of quality.

A “bug” is a synonym of a “fault”, which means “an event that occurs when the delivered service deviates from correct service”.

Quality is often considered by:

- Percentage of opened bugs in a period of time
- Percentage of resolved bugs in a period of time
- LOC of the committed fixes for a given bug

Feature Requests and Issues

Perfective Maintenance - fine tuning of all elements, functionalities and abilities to improve system operations and perfectness.

A “feature request” is a suggestion for how to improve or enhance the current software

We use the term “**issues**” to represent <u>both bugs and feature requests</u> of a system and “**issue summary**” to represent the description of an issue.

Prior Work

A large empirical study was conducted to validate the possibility of utilizing bug reports to examine how maintainability issues are expressed in open source software systems.

Used a large set of randomly sampled issue summaries from various open source systems

For each issue, an inspection team manually tagged it with one or more of the maintainability subgroup SQs or non-maintainability

We use the resulting set of issue-quality pairs relating to understandability as the ground truth data.

Examples of the issue-quality pairs

Issue Summary: Severe pauses/hang when dealing with large conversation backlogs

Quality tag: Scalability

Issue Summary: QuickSearch should be able to find a bug by its alias

Quality tag: Accessibility

Issue Summary: edit-multiple.html.tmpl uses &apos which Internet Explorer cannot use (patch included)

Quality tag: Portability

Issue Summary: Remove old integration tests from gaia system app

Quality tag: Diagnosability, Understandability

Prior Work

The approach provided insight into the maintainability of the systems, but required large manual effort to classify issue summaries.

During this process, we found that users are likely to use recurrent linguistic patterns in their sentences when reporting bugs or requesting new features.

Additionally, some linguistic patterns match the definitions found in standards and practice guidelines.

Example

One definition:

To users, software understandability is defined as the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use through **proper documentation and user guides**.

Sample issue summaries that express this concern

- Documentation does not mention you need to copy /media/js/app/local-settings.js-dist to /media/js/app/local-settings.js
- Update searchfox docs/ documentation
- Document the significance and usage of "popcorn" events
- Create a guided tour of Pontoon for users.
- Fix the lack of documentation for MediaDB.

Observation

If an issue summary contains “documentation”, then it is describing a understandability concern.

Since linguistic variables are central to fuzzy logic and the if-then formats match the definition of fuzzy rules,

Fuzzy rules are chosen to model the relationships between linguistic patterns and the classification results.

Fuzzy If-Then Rules

General format:

If x is A then y is B

Examples:

- If pressure is high, then volume is small.
- If the road is slippery, then driving is dangerous.
- If a tomato is red, then it is ripe.
- If the speed is high, then apply the brake a little.

An example of Fuzzy rules to classify vehicles

Given a vehicle V,

IF num_wheels = 4 AND has_engine = true AND drive_on_road = true THEN class is <u>automobile.</u>

IF num_wheels = 2 AND has_engine = false AND drive_on_road = true THEN class is <u>bicycle.</u>

…...

The Anatomy of Fuzzy rules

If *an issue summary contains certain linguistic patterns,* ***then*** *it is likely to express understandability concerns.*

Each fuzzy rule consists of two parts: a premise and a consequent.

The consequent determines the classification or the truth. The premise is the set of linguistic patterns.

“Fuzzy Hyperboxes”

The idea of “fuzzy hyperboxes” is adopted to learn rules from a set of examples by collecting data samples into "fuzzy hyperboxes".

If there are enough samples in one box, then a rule is formed.

Specifically, each rule is considered as one fuzzy hyperbox and rule weight is used to determine whether there exist enough samples for each rule.

Fuzzy Rules Classification

$$Rule\ R_k:\ IF\ x_1\ is\ A_{k1}\ AND\ ...\ AND\ x_n\ is\ A_{kn},$$

$$THEN\ class = C_k\ with\ CF_k,$$

where R_k is the kth fuzzy rule, x_1, ..., x_n are the linguistic variables, A_{ki} are the values used to the represent linguistic variables, C_k is a consequent class, and CF_k is a rule weight.

Rule weight => number of occurrence of the given rule R_k correctly classify in a given dataset with C_k

Rule Example

Understandability:

give_more = 1 && <N> ∈Understandability keyword: Issue summary has a semantic pattern of give_more and the noun belongs to the understandability keyword set.

Linguistic Patterns

- **Lexical Patterns**: words/phrases that occur in understandability related issue summaries with high frequency
- For example, when people report an issue that expresses **understandability** quality concern, they are more likely to mention words such as documentation or user manual.
- A set of keywords that contain the most frequent words was generated.
- When examining whether an issue summary contains any keywords, synonym comparison on the stemmed words/phrases are used.

Linguistic Patterns

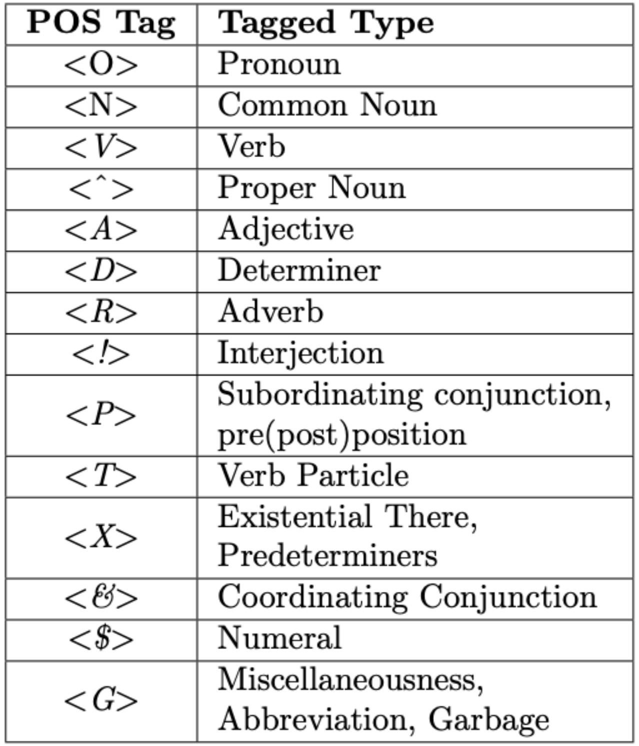
- **Syntax Patterns**: the specific sentence structures that frequently appear in understandability related issue summaries.
- For example, one of the syntax patterns is {start_vb = 1}, which means that the first word in the issue summary is a verb.
- POS Tags and Universal Dependencies are used to identify these patterns and find out the subject and main action of the sentence.

POS Tags

Part-of-speech (**POS**) taggers takes text as an input, and output the text with parts of speech assigned to each word/term, such as noun, verb, adjective, etc.

POS tagging is also referred to as word category disambiguation or grammatical tagging.

The goal is to identify each word's function in a given sentence.



POS Tag	**Tagged Type**
<O>	Pronoun
<N>	Common Noun
<*V*>	Verb
<ˆ>	Proper Noun
<*A*>	Adjective
<*D*>	Determiner
<*R*>	Adverb
<*!*>	Interjection
<*P*>	Subordinating conjunction, pre(post)position
<*T*>	Verb Particle
<*X*>	Existential There, Predeterminers
<*&*>	Coordinating Conjunction
<*$*>	Numeral
<*G*>	Miscellaneousness, Abbreviation, Garbage

Universal Dependencies

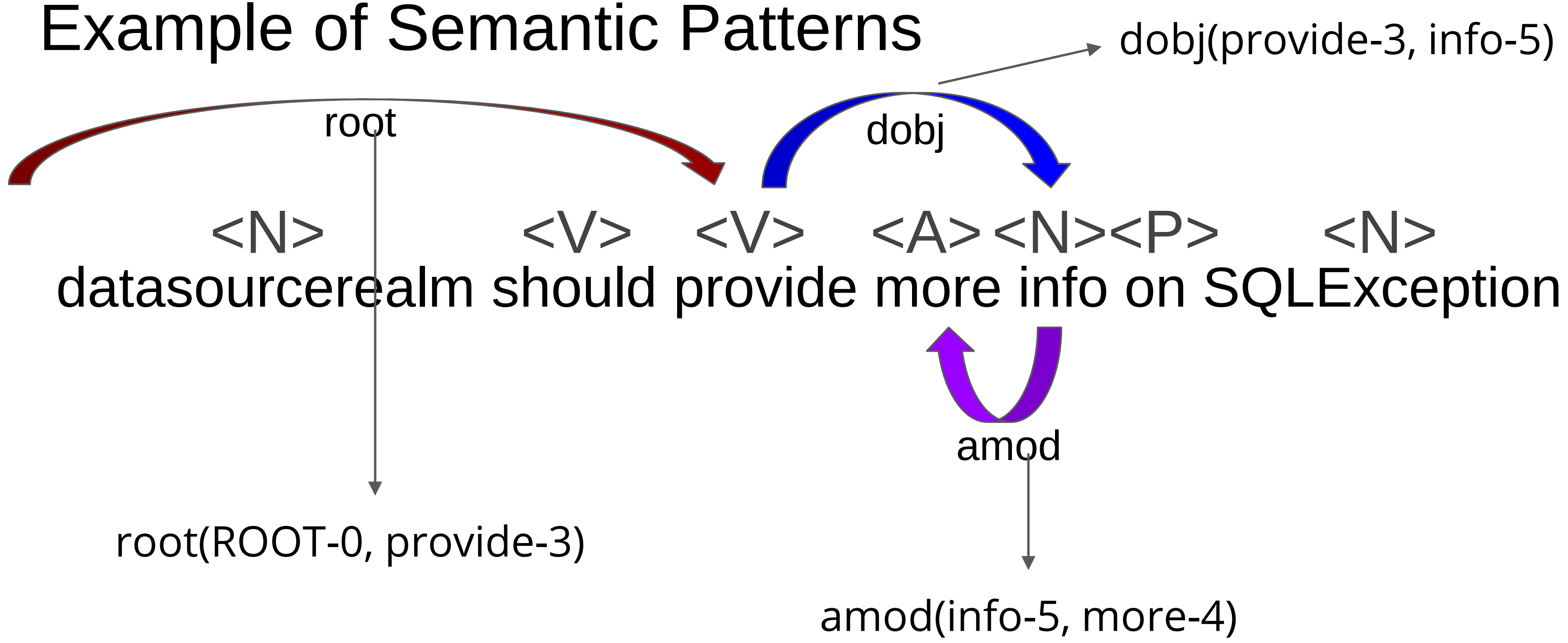
Universal Dependencies provide a representation for grammatical relationships for words in a sentence. They are used describe various syntactic relations.

For example, *nsubj*(nominal subject) describes the subject of a clause and identifies the ”do-er”. *cop*(copula) describes the relation of a function word used to link a subject to a nonverbal predicate.

Linguistic Patterns

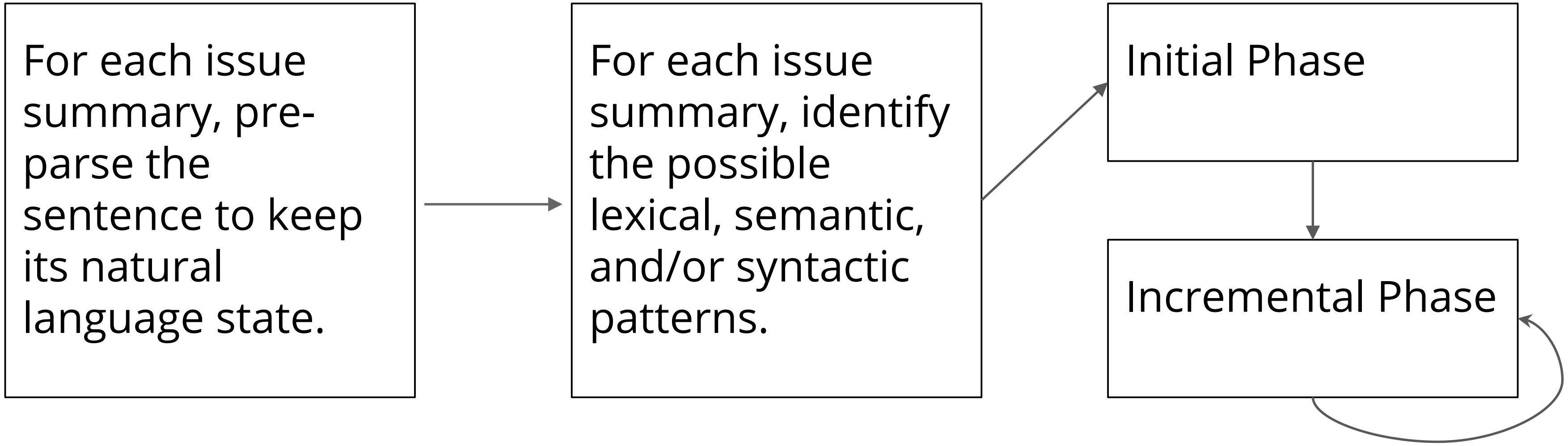
- Semantic Patterns: the meaning of linguistic expressions that frequently appear in understandability related issue summaries.
- For example, when people report an issue that expresses understandability quality concern, they are more likely to describe some parts of the system that needs more explanation or information.
- POS Tags and Universal Dependencies are used to identify these patterns.

Example of Semantic Patterns


dobj(provide-3, info-5)
root
dobj
<N> <V> <V> <A> <N> <P> <N>
datasourcerealm should provide more info on SQLException
amod
root(ROOT-0, provide-3)
amod(info-5, more-4)

$\{give\_more\} = 1 \longrightarrow$ understandability

Fuzzy Rule Generation


For each issue summary, pre-parse the sentence to keep its natural language state.
For each issue summary, identify the possible lexical, semantic, and/or syntactic patterns.
Initial Phase
Incremental Phase

The output of this phase is a set of stable fuzzy rules that can automatically tag an issue summary as relating to understandability

Pre-parse Issue Summaries

In order to improve the results of linguistic pattern identification, a set of pre-parsing steps are implemented to maintain the natural language state of the issue summaries.

- Source code, file path, URL are replaced with <CODE>, <FP> and <LINK> respectively.
- Trivial words such as hello, thank you, regards, etc. are replaced with <T>.
- Version numbers are replaced with <VERSION>.
- Component names are replaced with <COMP>.

Initial Phase

We first start with the set of initial rules $\{R_n\}$ identified from the definitions and practice guidelines as shown in the following list:

- Enhancement of the system => {action_”enhance”} = 1
- Adapting/porting to new circumstances => {actiona_”adapt”} = 1 AND succeeding <N> is <Version> OR <Sys_Name>
- Documentation, guide, user manual => {Documentation, guide, user manual} = 1
- Ease of understanding => {give_more} = 1, <A> is synonyms of {easier, better, nicer}

Incremental Phase

During this phase, we aim to improve the initial fuzzy rule set $\{R_n\}$ through incrementally classifying new issue summaries, and determining whether any new rules should be added to the existing rule sets.

In order to determine whether a rule should be added or the weight of a rule should be increased, we use correctness measurement, which is the average f-measure.

When any changes needs to be applied to the existing rule set, the change is only accepted when the correctness with the change is better than the current correctness.

Incremental Phase

This step is repeated until the performance of the current rule set is stable, which means either the correctness does not increase when adding in new rules or no new rules are being discovered.

As a result, a set of updated and stable fuzzy rules is obtained.

Evaluation of the Proposed Approach

- The rule quality is the most important evaluation criteria for rule extraction models and often it is measured through the accuracy of the model.
- The accuracy of extracted rules describes their ability to correctly classify data that is not used for the training of the model, which can be used to measure of the generalizability of the extracted rules.
- We use **accuracy, precision, recall, and f-measure** as the measurement to evaluate the performance of the proposed approach.

Correctness Measurement

Accuracy shows the percentage of correctly tagged issues in the entire dataset.

Precision indicates the correctly identified proportion of issues tagged as understandability.

Recall represents the coverage of correctly identified issues by the proposed approach.

F-Measure is the harmonic mean of precision and recall.

Results

We generated 23 rules to identify understandability related issue summaries: 4 from standards and practice principles and 19 from the dataset.

ID	Rule (Linguistic Patterns)	Type	Weight	Introduced in	Example of Issues
13	<V> <N> <P> <N> = 1 AND <V> ∊ {Key_Verb} AND {when, where, if} = 0 AND {neg} = 0	Syntax, Semantic, Lexical	92	{Im1}	Update DevHub front page to up-level partnership and simplify presentation.
19	<V> <N> = 1 AND <V> ∊ {Key_Verb}	Syntax, Semantic	35	{Im1}	Remove unused imports.
3	{documentation, guide, user manual} = 1	Lexical	10	{Rn}	[meta] Comprehensive Data Documentation Content.

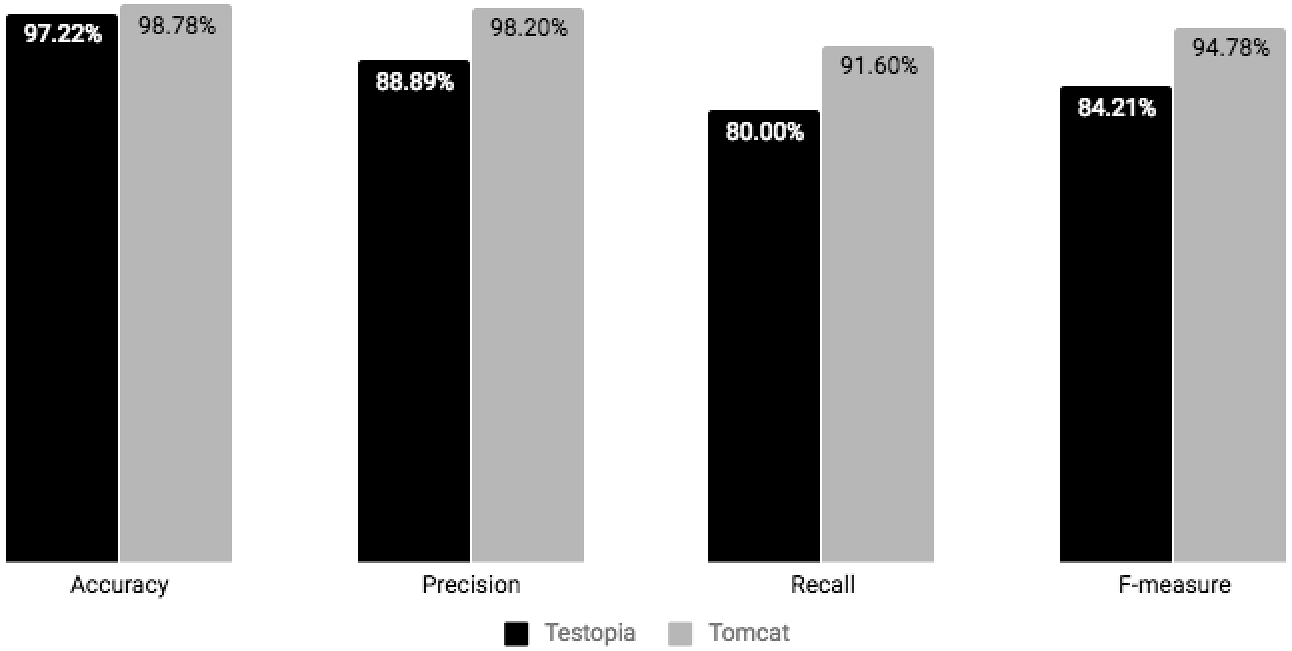
How effective are these at identifying understandability?

We manually tagged issue summaries from two projects not used for generating rules and compare with the results from the generated ruleset

We limit the study to only resolved and fixed issues since unfixed issues may be invalid and the quality concerns cannot be identified through issue summaries and follow-up discussions. Also issues containing typos are excluded.

Project	Domain	Earliest Issue Reported	Ecosystem	# of Issues
Tomcat 7	Web server	2009	Apache	985
Testopia	Testing tool	2006	Mozilla	431

Evaluation


97.22%
98.78%
88.89%
98.20%
80.00%
91.60%
84.21%
94.78%
Accuracy
Precision
Recall
F-measure
Testopia
Tomcat

Conclusions and Future Work

Our approach uses software artifacts generated throughout the development process to generate a set of fuzzy rules that can effectively identify understandability related issue summaries.

We have extended this approach for identifying other maintainability subgroup SQs

Investigate conflicts and synergies between SQs

Improve pattern extraction