Bug reports quality and potential room for improvement through automation.

Covered papers

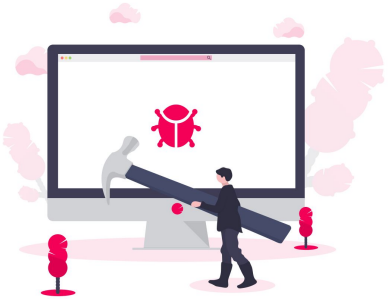
I. **What makes a good report? [BetJusSch08]**

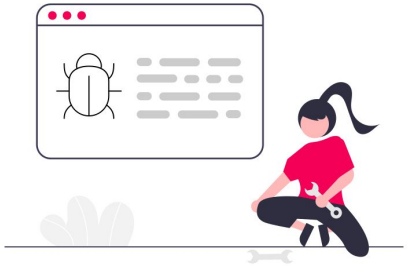
II. **BugListener: Identifying and Synthesizing Bug Reports from Collaborative Live Chats[ShiMuZha22]**

Disclaimer: **When presenting the various used techniques, some parts will only be explained at the surface level because of time frame & expertise limitations.**

Agenda

I. Introduction
II. What makes a good report?
    1. The Survey
    2. Rating bug reports
    3. CUEZILLA
III. Automated bug report generation: Collaborative Chat: BugListener
    1. Why collaborative live chat?
    2. BugListener's components
    3. Evaluation
    4. Potential use case
    5. Discussion





"The difference between a well-written bug report and a poorly written one can be the difference between a fix in the next release and a fix never."

Introduction

- A software bug report contains particular information about what is wrong with a software product and what needs to be resolved.

- Essential in software development
  - Help improve software quality and user experience.
  - Save cost and time by enabling debug at an earlier stage.
  - Maintain all stakeholders informed about the bug & assists them in taking corrective actions.
- Includes full description
- Sometimes includes stacktraces
- Bug report quality is crucial.
  - Poorly written bug reports slow developers down(e.g: Mozilla bug #109242) [BetJusSch08]
- Bug reports written by users:
  - Clear way of communication between users and developers
  - **Assumption:** Mismatch between what developers consider most helpful and what users provide.

What makes a good bug report ? [BetJusSch08]

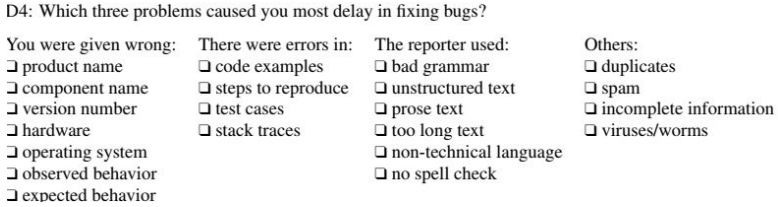
- Goals
  - Accurately define a good bug report
  - Improve bug reports quality

  ⇒ Bridge the gap between what developers think is most helpful and what users provide
- Survey on important parts of a bug report
  - Participants: 156 experienced developers & 310 experienced reporters across 3 projects: Apache, Eclipse and Mozilla
  - Developers rate chosen bug reports
- CUEZILLA: Measure bug reports quality & suggest improvements

1: MOZILLA bug #109242

The survey: Questions

- **Developers**
  - ***Which items have developers previously used when fixing bugs? Which three items helped the most?***
  - ***Which problems have developers encountered when fixing bugs? Which three problems caused most delay in fixing bugs?***

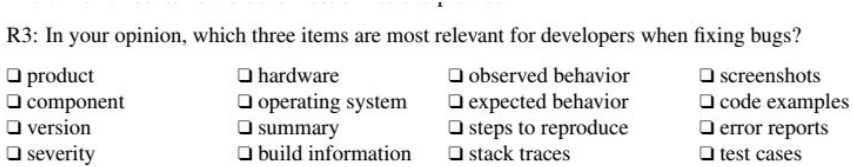


D4: Which three problems caused you most delay in fixing bugs?

You were given wrong:	There were errors in:	The reporter used:	Others:
❑ product name	❑ code examples	❑ bad grammar	❑ duplicates
❑ component name	❑ steps to reproduce	❑ unstructured text	❑ spam
❑ version number	❑ test cases	❑ prose text	❑ incomplete information
❑ hardware	❑ stack traces	❑ too long text	❑ viruses/worms
❑ operating system		❑ non-technical language	
❑ observed behavior		❑ no spell check	
❑ expected behavior			

[BetJusSch08, p3]

- **Reporters**
  - ***Which items have reporters previously provided? Which three items were most difficult to provide?***
  - ***Which three items do reporters consider to be most relevant for developers?***



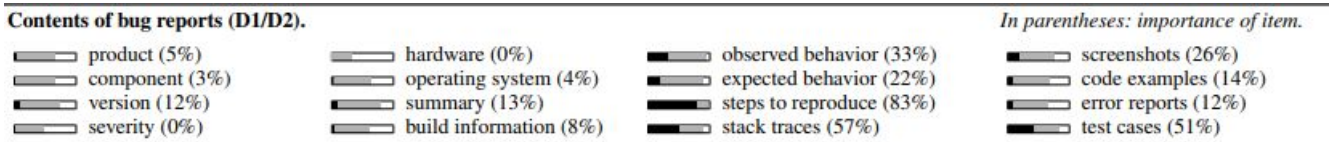
R3: In your opinion, which three items are most relevant for developers when fixing bugs?

❑ product	❑ hardware	❑ observed behavior	❑ screenshots
❑ component	❑ operating system	❑ expected behavior	❑ code examples
❑ version	❑ summary	❑ steps to reproduce	❑ error reports
❑ severity	❑ build information	❑ stack traces	❑ test cases

[BetJusSch08, p3]

The survey: Results

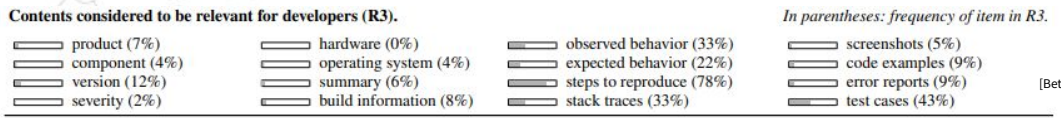
- **Developers:** *Which items have developers previously used when fixing bugs?* *Which three items helped the most?*



[BetJusSch08, p4]

  - **Importance of items**
    - Steps to reproduce > stack traces, test cases > observed behavior > expected behavior > code examples, summary, version, operating system, product, hardware.
- **Reporters**
  - **Items provided frequently by reporters: Expected behavior, observed behavior, steps to reproduce**
  - **Less frequently: stack traces, code examples, test cases**
  - **Most difficult items to provide:** Test cases > Steps to reproduce> Code examples > Stacktrace

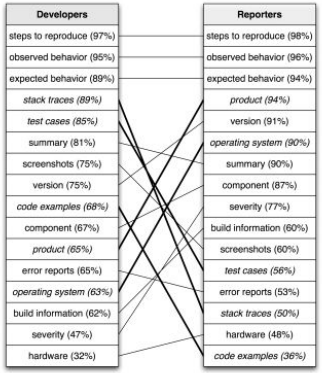
Which three items do reporters consider to be most relevant for developers?



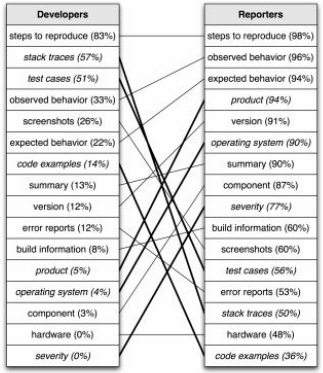
[BetJusSch08, p4]

The survey: Results

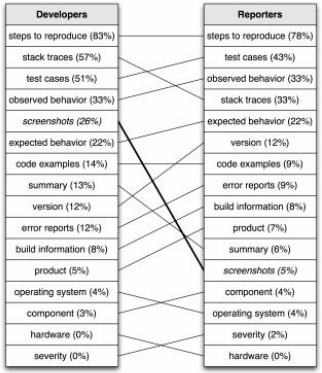
- **Do developers and reporters agree on important items in bug reports?**



(a) Information used by developers vs. provided by reporters.



(b) Most helpful for developers vs. provided by reporters.



(c) Most helpful for developers vs. reporters expected to be helpful.

[BetJusSch08, p5]

- **Comparison of**
  - **Information used by developers vs provided by reporters**
  - **Most helpful for developers vs provided by reporters**
  - **Most helpful for developers vs what reporters expect to be helpful**

Problems

- **“Which problems have developers encountered when fixing bugs? Which three problems caused most delay in fixing bugs?”** [BetJusSch08, p4]
  - Incomplete information, errors in steps to reproduce
  - Errors in test cases, incorrect versions number, observed behavior, expected behavior, bug duplicates
  - Fluency in the language
- **Additional problems**
  - Difference in knowledge levels
  - Violating netiquette
  - Complicated steps to reproduce
- **Some bug reports are favored**
  - Reports written by well known reporters
  - Reports where the reporter made the effort to identify the problematic code
  - Bugs with high severity

Rating bug reports

- Random sample of 100 bug reports from the respective projects
- Likert scale: very poor (1) → very good (5)
- Why?
  - Verify the survey's results with real-world examples.
  - Rating scores will be used to evaluate CUEZILLA
- But
  - Can't ratings be subjective?
  - Can developers agree on rating by chance?

⇒ Compute standard deviations of quality ratings

- Low standard deviation across 92% of bug reports
- Developers generally agree on the quality of bug reports

5/5 rating score

Run the following example. Double click on a tree item and notice that it does not expand.

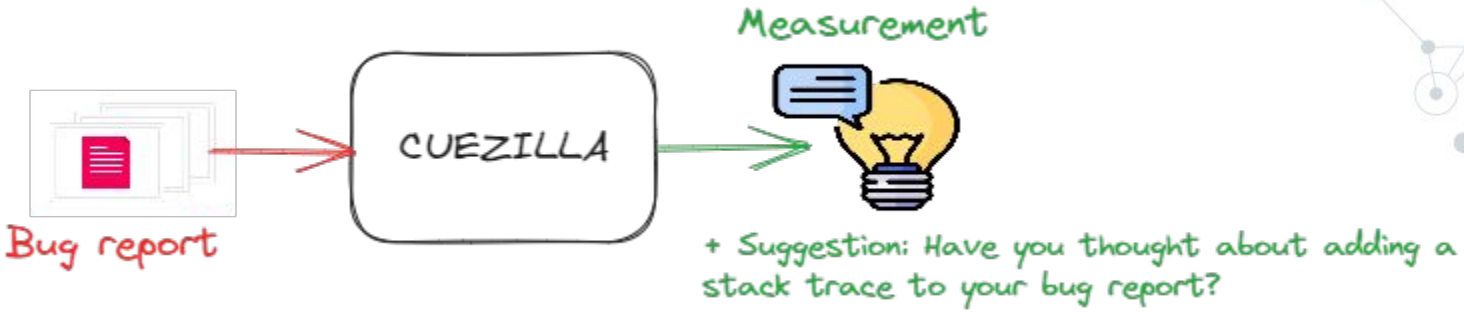
Comment out the Selection listener and now double click on any tree item and notice that it expands.

```
public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    [...] (21 lines of code removed)
    display.dispose();
}
```

[BetJusSch08, p6] (ECLIPSE bug report #31021)

⇒ It is possible to build a tool that learns from bug reports to measure quality of new bug reports ⇒ CUEZILLA

CUEZILLA



- Measure bug reports quality based on its content
- Based on the survey, CUEZILLA computes quality score of bug reports
  - Binary: e.g: is screenshot present ?
  - Continuous: e.g: readability
- Completeness of a bug report
  - NLP operations to identify keywords
    - remove stop-words → Stemming → Select words present in at least 1% of bug reports
  - Assign keywords to groups:
    - action items (e.g., open, select, click)
    - expected and observed behavior (e.g., error, missing)
    - steps to reproduce (e.g., steps, repro)
    - build-related (e.g., build)
    - user interface elements (e.g., toolbar, menu, dialog)
- Analyze attachments: Code samples, stack traces, patches, screenshots

CUEZILLA: How are recommendations generated ?

- CUEZILLA delivers useful random facts that are statistically scraped from bug databases
- How is this done ?
  - Sample 50.000 bug report from each project
  - A bug has resolution state: [FIXED], [DUPLICATE], [MOVED], [WONTFIX], [WORKSFORME]
  - Compute the test results to know whether the presence of a certain feature significantly determine the resolution category of a bug.
- What items in a bug report shorten its life time(gets fixed!)?
- Findings from the 3 projects(sample size: 50.000 x 3)
  - Bug reports including stack traces are resolved more quickly(Across 3 Projects).
  - Bug reports that are more readable have shorter lives(Across 3 Projects).
  - Including code examples in your bug report enhances the likelihood of it being fixed.(MOZILLA)

Evaluation

- Supervised learning models : support vector machines (SVM), generalized linear regression (GLR), and stepwise linear regression
- Evaluation
  - Within the same project: For a given project A, predict quality of a bug report within A using the other bug reports in A. (leave-one-out cross-validation)
  - Across projects. Use model built from all rated bug reports of project A, and apply it to predict the quality of all rated bugs in project B.
- Prediction models perform comparably well
- Models trained from one project can be applied to other projects without losing much predictive power

⇒ CUEZILLA models are portable across different projects but they perform best within the same project

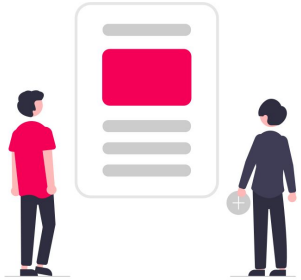
- CUEZILLA can measure quality of bug reports within reasonable accuracy
- CUEZILLA has potential to be integrated in bug tracking systems

Threats to validity

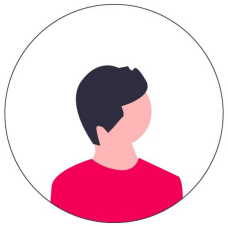
- Selection of developers/reporters
- Self selection principle: Participation in the survey is voluntary
- Time constraint hinders completeness
- Generalization: What about closed-software projects?

Conclusion

- What does this paper achieve?
  - Provide a scientific evidence to common-sense good practices.E.g: Stack traces are helpful
- Steps to reproduce and stack traces are the most useful elements of a bug report.
- The most serious issues that developers face include errors in steps to reproduce, incomplete information, and wrong observed behavior
- Bug duplicates are encountered often but aren't considered harmful
- Mismatch between what information developers consider as important and what users provide
- CUEZILLA
  - Rate up to 41% bug reports in complete agreement with developers.
  - Present recommendations to improve bug report quality



Mismatch between what information developers consider as important and what users provide



Why collaborative live chat?

Live chatting is more efficient compared to asynchronous communication such as emails or forums. [LinZagD.SSer16][ShiMinE.H09]

⇒ It is becoming an essential part of most software development processes

Chat conversation includes:

- Unexpected Behaviors
- Development problems
- User Feedback
- Social Events

Why collaborative live chat?

Live chatting is more efficient compared to asynchronous communication such as emails or forums. [LinZagD.SSer16][ShiMinE.H09]

⇒ **it is becoming an essential part of most software development processes**

Chat conversation includes:

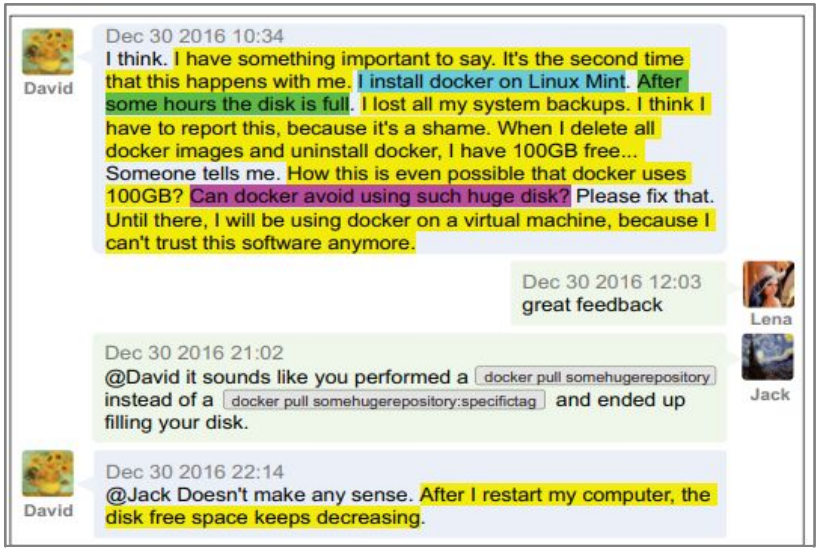
- **Unexpected Behaviors** ⇐ 32% of chat dialogs are reporting unexpected behavior [Shi et al.]
- **Development problems**
- **User Feedback**
- **Social Events**

Use-case: Coordination between developers

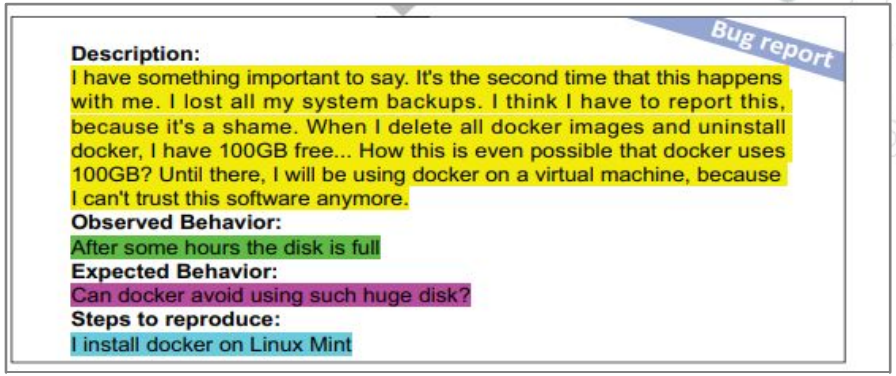
- Approach:
    1. Discuss bug in Slack, Gitter,MS Teams...
    2. Open project management software
    3. Create an issue(Task type: Bug)
    4. Write bug report

- A bit more sophisticated method: Jira for Slack **Integration**

Automated approach using BugListener



Live chat dialog extracted from docker's Gitter [ShiMuZha22]



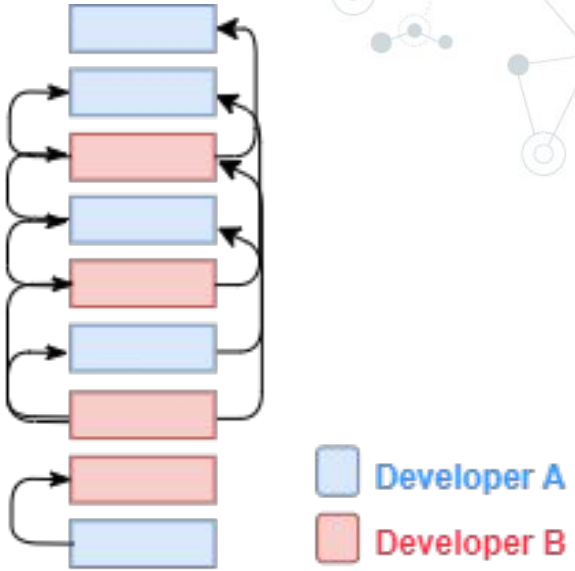
Output produced by BugListener [ShiMuZha22]

Challenges

- Noisy chat conversations
  - Off-topics, irrelevant informations
- Entangled chat conversations
  - Context dependent utterances
- Insufficient labeled resource
  - Data annotation through human intervention
- Quality of produced bug-report
  - Description, observed behavior, expected behavior, steps to reproduce

Dialog Disentanglement

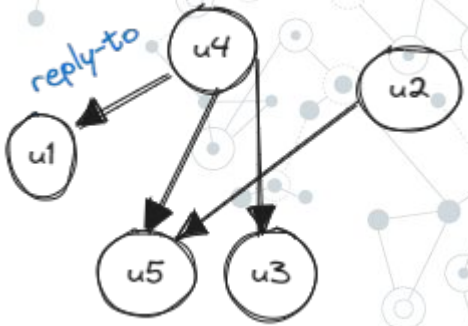
- Motivation: Interleaved conversations need to be split
- Goal of disentanglement:
    - Establish a “reply to” relationship between utterances
    - Cluster utterances as one dialog
- Given chat log L: f(L) disentangle it into separate dialogs {D1,D2,D3...}
- Experiment with various disentanglement models(FF, BERT, E2E...)
- Feed-Forward model yields best results



Data augmentation

- Motivation: Issues with data imbalance and limited annotation
- Data augmentation
  - Dialog mutation while keeping semantics
    - Long utterance: Word level replacement
    - Short utterance: Utterance level replacement
- Data balance
  - Augment Bug-report dialogs
  - Match the number of bug-report dialogs to non bug-report dialogs

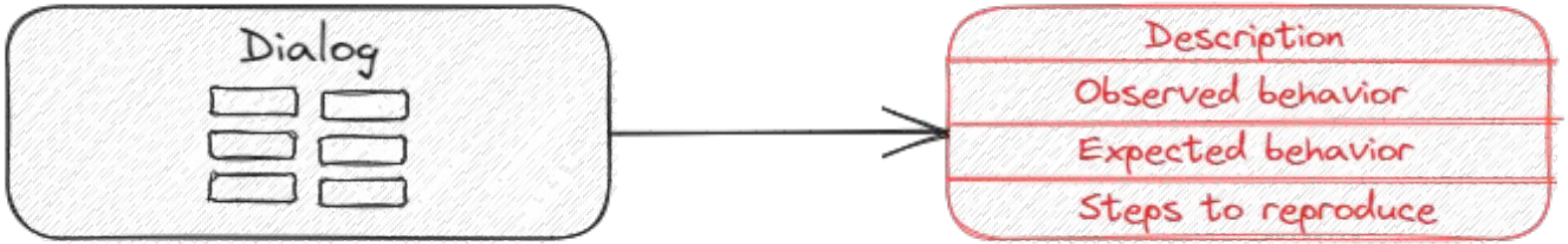
Bug-report identification(BRI)



- Motivation: Identify bug report dialogs from separated dialogs
- Binary function $f(D_i) \implies \{true, false\}$
- Utterance embedding:
  - Word encoding : Encode semantic information of words
  - Utterance encoding: Learn the representation of utterances
- Graph-based Context embedding: capture the graphical context of utterances in one dialog
  - Use the “reply-to” relationship between utterances in a dialog to build a directed graph
  - Embed dialog graph context
- Use the obtained representation of an entire dialog to classify it as either a positive or a negative bug-report dialog.

Bug-report synthesis(BRS)

- **Motivation: Synthesize the bug reports from predicted bug report dialogs**



- **Challenge: high volume of live chat data & limited labeled data ⇒ low volume training data for bug report synthesis task.**
  - **Solution: twice fine-tuned BERT model**
- **BERT: Pre-trained on large amounts of text (Wikipedia:2500M words, BookCorpus(800M words)**

Evaluation: Methodology

- Selected OSS communities:
  - Top-1 most participated communities from six active domains
    - Front end framework : Angular
    - Mobile: Appium
    - Data science: DL4J
    - DevOps: Docker
    - Collaboration: Gitter
    - Programming Language: TypeScript
  - Use Gitter as communication tool
- Data preprocessing & disentanglement
- Sampling
  - Random 100 dialogs from each OSS community
    - Only 1.1% of the population. **Problematic ?**
- Filtering: exclude noisy dialogs

Evaluation: Methodology

- Labeling
  - Manually correct disentanglement results
  - Manually correct the “reply-to” relationship
  - Manually label dialogs with BR(Bug Report) or NBR(Not BugReport)
  - Manually label individual sentences with OB, EB and SR
  - **Validity?**
    - Agreement between labelers:
      - 79 % correctness of automated dialog disentanglement
      - Average Cohen’s Kappa(BRI) = 0.87
      - Average Cohen’s Kappa(BRS) = 0.84
- Balance BR and NBR data
- Include an external dataset for transfer learning

BugListener vs state-of-the-art baselines

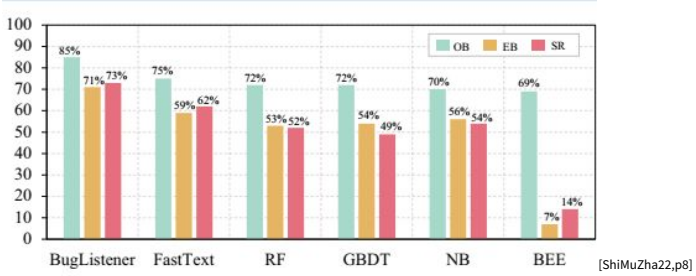
“ How effective is BugListener in identifying bug-report dialogs from live chat data?”

Table 2: Baseline comparison across the six communities for bug-report dialog identification (%).

Methods	Angular			Appium			Docker			DL4J			Gitter			Typescript			Average		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
BugListener	82.93	79.07	80.95	69.39	80.95	74.73	77.42	78.69	78.05	85.07	72.15	78.08	82.09	87.30	84.62	70.00	70.00	70.00	77.82	78.03	77.74
NB	58.88	73.26	65.28	62.22	66.67	64.37	65.52	31.15	42.22	62.79	34.18	44.26	72.92	55.56	63.06	35.29	30.00	32.43	59.60	48.47	51.94
GBDT	72.22	60.47	65.82	65.17	69.05	67.05	66.00	54.10	59.46	85.00	64.56	73.38	59.77	82.54	69.33	35.14	65.00	45.61	63.88	65.95	63.44
RF	75.00	59.30	66.23	72.15	67.86	69.94	68.75	36.07	47.31	72.73	20.25	31.68	62.34	76.19	68.57	60.00	30.00	40.00	68.50	48.28	53.96
FastText	77.59	52.33	62.50	68.54	72.62	70.52	56.60	49.18	52.63	74.51	48.10	58.46	67.24	61.90	64.46	40.91	45.00	42.86	64.23	54.86	58.57
CNC	80.36	52.33	63.38	67.05	70.24	68.60	74.51	62.29	67.86	84.44	48.10	61.29	68.18	71.43	69.77	52.00	65.00	57.78	71.09	61.57	64.78
DECA	51.32	45.35	48.15	51.16	52.38	51.76	45.57	59.02	51.43	42.22	48.10	44.97	55.36	49.20	52.10	21.57	55.00	30.99	44.53	51.51	46.57
Casper	67.65	53.49	59.74	66.06	85.71	74.61	60.56	70.49	65.15	82.14	58.23	68.15	73.33	69.84	71.54	32.50	65.00	43.33	63.71	67.13	63.75

[ShiMuZha22,p7]

How effective is BugListener in synthesizing bug reports?



[ShiMuZha22,p8]

Fig. 3: Baseline comparison for bug report synthesis.

Human evaluation: Procedure

- Apply BugListener to 5 **new** communities : Webdriverio, Scala, Materialize, Webpack, Pandas
- Human annotators
  - 2 PhD students
  - 2 Master students
  - 3 Professional developers
  - 2 Senior researchers
- Scrape recent live chats(3443 utterances) → Disentanglement(562 dialogs) → 31 potential bug reports identified
- Each bug report is evaluated by 3 annotators
- Evaluate based on a survey [ShiMuZha22,9]
  - Correctness: Whether the dialog is discussing a bug that should be reported at that moment?
  - Quality: How would you rate the quality of Description, Observed Behavior, Expected Behavior, and Step to Reproduce in the bug report?
  - Usefulness: How would you rate the usefulness of BugListener?

Human evaluation: Results

- Correctness: 77% of identified bug reports are correct
- Quality
  - 83% ~ satisfaction on description
  - **Acceptable** satisfaction on EB, OB, SR
- Usefulness: 71% agreement that BugListener is useful

HOW CAN IT BE USEFUL?

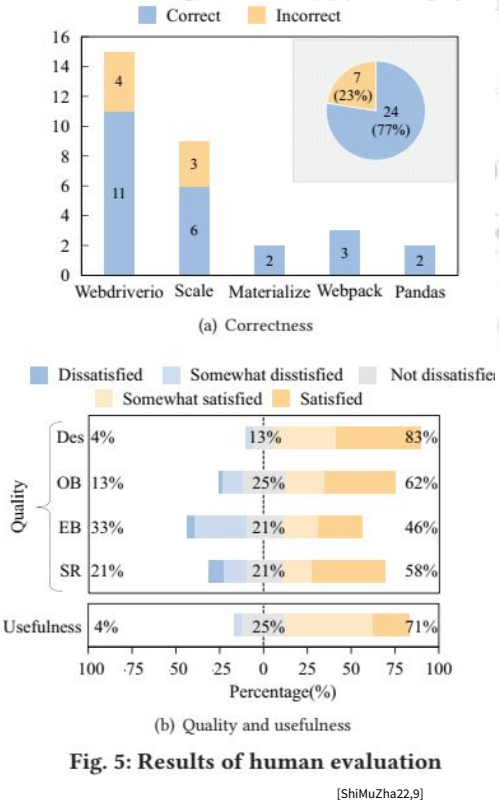


Fig. 5: Results of human evaluation

[ShiMuZha22,9]

Potential use-case

- Semi-automated
- Scenario
  - OSS repository owner/core team members subscribe to adequate chat rooms via BugListener
  - BugListener monitors the chatrooms and notifies subscribers about potential bug reports periodically
  - Subscribers confirm that it is actually a bug and assess quality
  - BugListener automatically create issues on code repositories

- Problem: Relying only on BugListener for Bugs report identification is risky because of no perfect recall(77%). What if a dialog is a bug but doesn't get identified as such?
- Is the tradeoff worth it ?

Discussion

- How is the quality of automatically created bug reports ? ✔
- Why do developers discuss bugs in chat instead of creating a proper bug report?
  - Better collaboration due to real-time communication
  - Initial assessment: Evaluate bugs severity and impact
  - Fast-paced development environments where the goal is to find quick fixes and documentation is sometimes unnecessary.
  - Informal communication → More freedom
- Does the bug report need to be proper?
  - Minimalist approach where only necessary items are present
  - Minimalist bug report: short term high risk, long term chaos
    - Insufficient information → increased back-and-forth communication
    - Misunderstood severity and impact → faulty prioritization
  - Bug trackers make it easier to create **proper** bug reports through standardized format & required fields.
- What can be a potential solution to the problem?
  - Semi-automated BugListener integration for chat clients & BugTrackers ✔

Thank you for your time !
&
Happy debugging !

Credits

- What makes a good report? [BetJusSch08]
- BugListener: Identifying and Synthesizing Bug Reports from Collaborative Live Chats[ShiMuZha22]
- Why Developers Are Slacking Off: Understanding How Software Teams Use Slack? [LinZagD.SSer16,333-336]
- On the Use of Internet Relay Chat (IRC) Meetings by Developers of the GNOME GTK+ Project [ShiMinE.H09,107–110]
- On the Use of Internet Relay Chat (IRC) Meetings by Developers of the GNOME GTK+ Project [ShiMinE.H09,147-156]
- Illustrations: undraw.co
- Presentation theme: https://www.slidescarnival.com/cordelia-free-presentation-template/216

Credits

- What makes a good report? [BetJusSch08]
- BugListener: Identifying and Synthesizing Bug Reports from Collaborative Live Chats[ShiMuZha22]
- Why Developers Are Slacking Off: Understanding How Software Teams Use Slack? [LinZagD.SSer16,333-336]
- On the Use of Internet Relay Chat (IRC) Meetings by Developers of the GNOME GTK+ Project [ShiMinE.H09,107–110]
- On the Use of Internet Relay Chat (IRC) Meetings by Developers of the GNOME GTK+ Project [ShiMinE.H09,147-156]
- Illustrations: undraw.co
- Presentation theme: https://www.slidescarnival.com/cordelia-free-presentation-template/216