Understanding the Issue Resolution Process of Mozilla Firefox

1 Research Goal

We are conducting a study to determine how developers resolve issues (defects, tasks, enhancements, feature requests, etc.) for different problem kinds, and how they document the issue resolution process in issue reports. We will focus on issue reports of Mozilla Firefox (the open-source web browser).

To that end, we will annotate the content in issue reports (i.e., their title and comments) to identify the stages developers perform to solve issues. The annotations will be used to derive common patterns of issue resolution.

2 Task Summary

In this task, you will **annotate N issue reports from Mozilla Firefox**, following an open coding methodology. You will annotate issue comments and titles using an existing catalog of *codes* (aka. *annotations*). The minimal unit of annotation is complete sentences in issue comments; however, you can annotate multiple sentences or paragraphs, or even entire comments if the **textual snippets**, as a whole, describe information relevant to the issue resolution process. The catalog (*annotation_codes* sheet) is expected to change during the coding/annotation process by you and other annotators. You will also identify the problem categories developers describe in issue reports. This categorization is also documented in a catalog (*problem_categories* sheet), which can also change during the annotation process.

In the *issues* sheet (link given below), you will find the list of issues assigned to you. You will annotate the web pages of the issues, using the annotation tool *Hypothesis*.

The project lead will be the main point of contact for this task.

3 Software System

The coding focuses on the issues of two “products” of Mozilla Firefox (as defined in Firefox’s issue tracker):

- **Core**: This product includes shared core components used by Firefox and other Mozilla software, including handling of Web content; Gecko, HTML, CSS, layout, DOM, scripts, images, networking, etc.
- **Firefox**: This product includes Firefox user interface issues: in menus, bookmarks, location bar, and preferences.

4 Overall Procedure

This is the overall procedure for the annotation task:

1. You will read this document and get familiar with the annotation tool, existing codes, rules of annotation, etc.
2. You will attend to a training session where two trainer will demonstrate the task and how to annotate issues with example. They also discuss if you have any questions.
3. You will annotate the first 5 issues according to the guidelines of this document and the rules defined in the annotation catalog.
4. Project lead will review your annotations and make comments on them.
5. You and project lead will meet to discuss the comments. Project lead will give you feedback and discuss disagreements to reach a consensus.
6. You will annotate the remaining N-5 issues.
7. Project lead will review your annotations and make comments on them.
8. You and project lead will meet to discuss the comments. Project lead will give you feedback and discuss disagreements to reach a consensus.

5 Resources Needed for Coding

We use a Google spreadsheet to keep track of the issues that are annotated. It also contains catalogs of codes and problem categories.
The spreadsheet is named as: ***issue_annotation***

The spreadsheet contains several sheets. You will use **three sheets**:

Sheet	Description	Usage
issues	Contains the list of issues to be annotated.	You will update this sheet while annotating issues.
annotation_codes	Contains the annotation codes with descriptions, rules, and examples.	You need to read this sheet before starting the annotation. You may need to use this as a reference or update while doing annotations.
problem_categories	Contains problem categories with descriptions and examples.	You need to read this sheet before starting the annotation. You may need to use this as a reference or update while doing annotations.

During the annotation process, you can consult the documentation of Mozilla about handling issues/bugs: https://firefox-source-docs.mozilla.org/bug-mgmt/index.html
Useful resources on that website are:

- Bug types (aka issue types):
    - https://firefox-source-docs.mozilla.org/bug-mgmt/guides/bug-types.html
- Issue priority and severity:
    - https://firefox-source-docs.mozilla.org/bug-mgmt/guides/priority.html
    - https://firefox-source-docs.mozilla.org/bug-mgmt/guides/severity.html
- Issue triaging:
    - https://firefox-source-docs.mozilla.org/bug-mgmt/policies/triage-bugzilla.html

IMPORTANT: all tasks, defects, enhancements, and other change requests for Firefox are documented in Firefox's issue tracker (based on Bugzilla). No other systems are used by Firefox to this end. However, supporting systems are integrated into the issue tracker (version control system, code review system, static analysis tools, etc.). For more information, see this web page: https://firefox-source-docs.mozilla.org/bug-mgmt/guides/bug-pipeline.html

6 Glossary

Issue-related terminology:

- **Issue Report (aka issue):** a web document used to keep track of tasks, defects, enhancements, and other change requests. Issues are web pages in the issue tracker.
- **Issue title:** a short description of the issue.
- **Issue comment:** a message a person or a bot makes on the issue.
- **Issue description:** the first k comments on the issue made by the reporter.
- **Patch:** a code change (i.e., a diff) to implement a solution to the reported problem in the issue.
- **Attached patch:** a patch attached to an issue.
- **Commit:** the set of code changes recorded in the version control system (aka a code repository).
- **Pull request:** a request to merge a patch (a code change) from one branch into another within a code repository.

Terminology related to the annotation process:

- **Code:** a tag used to annotate issue textual snippet. The tag has a specific meaning and rules that indicate how and when to use the tag.
- **Annotation:** a piece of text (aka, textual snippet) in the issue annotated with **one or more codes**. Only complete sentences, phrases within sentences, or paragraphs are annotated.
- **Hypothesis:** the webpage annotation tool used to annotate issue content.
- **Problem:** the problem that is described in the issue which needs to be resolved.
- **Solution:** a potential solution proposed to address or solve the problem. An issue could have multiple solutions – eventually one is implementation.
- **Solution implementation:** the implementation of the solution in the software, typically via code changes (i.e., patches/commits).
- **Problem Category:** the type of problem or task reported in an issue.
- **Problem Class:** the topic where a problem category falls into.

7 Installing the annotation tool

To install the Hypothesis tool:

1. Add the Hypothesis extension to your browser:
    a. Chrome: https://chrome.google.com/webstore/detail/hypothesis-web-pdf-annota/bjfhmglciegochdpefhhlphglcehbmek
    b. Firefox: https://addons.mozilla.org/en-US/firefox/addon/hypothes-is-bookmarklet/
    c. Edge: https://web.hypothes.is/help/installing-the-hypothesis-chrome-extension-in-microsoft-edge/#:~:text=Install%20the%20extension%20in%20Edge,Add%20extension%E2%80%9D%20in%20the%20popup.
2. Log into Hypothesis (on your web browser) and the project group
3. Learn how to use hypothesis by watching this short video:
    a. https://www.youtube.com/watch?v=IsUD4UrUEHE

8 Getting familiar with the catalogs of codes and issue categories

Complete the following steps:

1. Read and understand the annotation codes with their descriptions, rules, and examples from the *annotation_codes* sheet.
2. Read and understand the problem classes and categories with their descriptions and examples from the *problem_categories* sheet.
3. Feel free to ask the project lead any questions about codes or problem categories.

9 Annotating the Issues

The following steps should be followed for each issue assigned to you. Feel free to ask the project lead any questions about the issue content or the procedure to annotate the issues.

The annotation process essentially comprises two major steps:

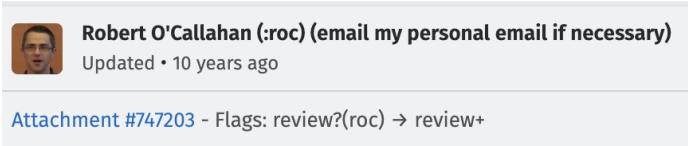
1. Identify the category of the problem reported in the issue
2. Annotate issue content (title and comments)

These steps are detailed below.

9.1 General recommendations

1. Annotate the issues in **30-minute sessions**, taking breaks between sessions. Please do not code the issues for a long time.

2. Annotate meaningful sentences (grammatically complete phrases as people write them) in the comments. You can annotate complete sentences, phrases within sentences, or paragraphs **with one or more codes.**
3. Do not annotate the metadata comments in the issues. Annotate only the issue title, description, other comments (human comments or bot messages), and attached patches.
   Example of metadata comments:



4. If you think the textual spinnet does not fit the definition of the codes in the *annotation_codes* sheet, then:
   a. Feel free also to discuss this situation with the project lead
   b. You can add a new code to the catalog with a description, rule, and example for that comment in the sheet.
   c. You can also adapt the definition of an existing code after you consult with the project lead
5. Feel free to read multiple times the comments on the issue to make sure you understand their meaning.
6. You can add any comment in the *comments* column of the *issues* sheet.
7. If you find that a comment is hidden in the issue, then expand that comment and annotate it with the appropriate code.
8. You do not need to annotate every issue comment. We have defined a set of rules for the cases that do not require annotation in *annotation_codes* sheet.
9. You will annotate using the **“<group-name>”** group in Hypothesis.

9.2 Identifying the problem category

1. Go to the *issues* sheet and open an issue by clicking the URL.
2. Make a first pass to the issue to understand the overall concept of the issue: read the issue title and comments.
   a. You can re-read the issue at any time
3. If the issue is a **pull request**, then replace 0 with 1 in the *is_pull_request?* column in the *issues* sheet. If the issue is filed for **tracking a merge**, or a **placeholder of another issue**, then add ‘merge’ or ‘placeholder bug’ in the *is_pull_request?* column. In these cases, there is no need to annotate the issue any further and continue with the next one.
4. Identify the problem category and add them to the *issues* sheet by selecting an option from the drop-down menu.
   a. Feel free to use Google or ChatGPT or consult Firefox’ documentation to clarify concepts or the problem.

    b. If you use ChatGPT, make sure you double-check its answers (via Google searching or consulting Firefox's documentation) its responses as it may not give a correct response.
5. If the problem falls into the class/category that is already found in the drop-down list, select it from the menu.
6. Otherwise, create a class/category and add a row with a description and a real example (i.e., from the issue) in the *problem_category* sheet. You will see the item instantly in the drop-down list in the *issues* sheet.

9.3 Annotating the issue content

1. Identify the comment that is related to the issue resolution process.
2. Read the comments one by one thoroughly and annotate with the specific codes that best describe the comments.
    a. Distinguish different kinds of requests (see the codes about requests in the *annotation_codes* sheet)
    b. Identify the proposed potential solution and solution review
    c. Identify the implementation of the solution (attached patch, attached file, try push, review commit, review request updated, etc.)
        i. The attached patches indicate a solution implementation.
        ii. If there is a meaningful comment (related to the solution/implementation) with the attached patch, annotate the comment as CODE_IMPLEMENTATION. (Note: we may need to annotate the comment or a part of the comment with multiple codes based on the provided information).
        iii. If there is no comment with the attached patch, annotate the full text starting with "Attached patch" or "Attached file".
        iv. There can be multiple attached patches sequentially. In that case, annotate all the patches separately.
    d. Identify content related to code reviews, solution verifications, and other codes if they are present in the comments and annotate them accordingly.
    e. For code reviews, there can be multiple sub-comments inside a single code review. Annotate all the sub-comments as CODE_REVIEW. If there are questions among the sub-comments, then annotate that question as a QUESTION. Hence, for the question in a code review, there can be 2 annotations.

```
>+
>+  nsCOMPtr<nsIDOMEventTarget> target(do_QueryInterface(piTarget));
>+  // XXX Why the else case of this condition isn't error?
```

```
Same here. Just return some error code if !target
```

```
> NS_IMETHODIMP
> nsEditorEventListener::KeyDown(nsIDOMEvent* aKeyEvent)
> {
>+  NS_ENSURE_TRUE(mEditor, NS_ERROR_NOT_AVAILABLE);
>   return NS_OK;
> }
```

```
No need for NS_ENSURE_TRUE check here.
```

```
>
> NS_IMETHODIMP
> nsEditorEventListener::KeyUp(nsIDOMEvent* aKeyEvent)
> {
>+  NS_ENSURE_TRUE(mEditor, NS_ERROR_NOT_AVAILABLE);
>   return NS_OK;
> }
```

```
No need for NS_ENSURE_TRUE check here.
```

f. Some comments are created by bots (BugBot, PulseBot, etc.), but may include human-written comments.
    i. BugBot: people use this bot to automate various tasks, such as updating the issue status, posting comments, assigning bugs to developers, or sending out notifications.
        1. Some BugBot comments contain a message from a person. If this is the case, annotate it with specific codes. Ex: https://bugzilla.mozilla.org/show_bug.cgi?id=1809080
        2. If the comment contains no useful information, do not annotate that. Ex: https://bugzilla.mozilla.org/show_bug.cgi?id=1809080
    ii. PulseBot: Used to post the commit message with the link to the commit. That means this is a CODE_IMPLEMENTATION.
        1. If you already coded a previous attached patch/attached file as CODE_IMPLEMENTATION and the PulseBot commit message is the same as the attached patch, do not code the PulseBot message again. Ex: https://bugzilla.mozilla.org/show_bug.cgi?id=1516605
        2. If there is no attached patch or you did not code any CODE_IMPLEMENTATION before the PulseBot commit message or the PulseBot commit message is different from a previous CODE_IMPLEMENTATION, code the PulseBot commit message as CODE_IMPLEMENTATION. Ex: https://bugzilla.mozilla.org/show_bug.cgi?id=1179123

g. If you find a SOLUTION_VERIFICATION without any prior CODE_IMPLEMENTATION, investigate why this happened. Perhaps there is an attached patch with the problem description. Mark this issue with red color in the *issues* sheet and add a comment in the *comments* column.

3. Don't get confused between REPRODUCTION_ATTEMPT and SOLUTION_VERIFICATION.
    a. REPRODUCTION_ATTEMPT: this is about trying to reproduce the issue (by running the code) after the issue is reported and before the solution is implemented.

b. SOLUTION_VERIFICATION: this is about trying to reproduce the issue (by running/testing the code) to verify if the solution implementation solved the issue. Here, issue reproduction is conducted after the implementation of the solution.

9.4 Finalizing the annotation

1. Revise all the annotations from top to bottom for a sanity check and update if necessary.
2. Copy the Hypothesis annotation URL by clicking the *share* option. Add the link to the *annotation_link* column in the *Issues* sheet.
3. Replace 0 with 1 in the *coded?* column in the *issues* sheet to indicate you have coded that issue.
4. Once again, at any moment during the coding procedure, feel free to add any comment in the *comments* column of the *issues* sheet.

10 Reviewing the Annotations

10.1 Preparation

1. Install hypothesis tool and create an account (Section 7)
2. Join <*group-name*> hypothesis group
3. Get familiar with the catalogs of codes and issue categories (Section 8)
4. Read and understand the issue annotation procedure (Section 9)

10.2 Review

For reviewing the annotations, perform the following steps for each issue:

1. Open the issue assigned to you by using the given *annotation_link* from the *issues* sheet. On the webpage of the issue, you will review the annotations made under <*group-name*> group.
2. Read and understand the issue title and comments one by one and check the annotations. For checking the annotations, follow the same guidelines as Section 9.
    a. If you agree with the annotations then keep them as they are.
    b. If you disagree with the annotations, make a reply to that *hypothesis* annotation stating your concern.
        i. For replying always use **lower-case** letters. Do NOT start with an **upper-case** letter.
        ii. Be critical of your judgments. Do not hesitate to disagree.
3. Based on the issue title and the comments, categorize the issue and check the *issue_category* from the *issues* sheet.
    a. If you agree with them, keep them as they are.
    b. If you disagree, then write your comment in the *comments* column.
4. Typical cases where you may disagree:
    a. The current code is incorrect
    b. Another code is more suitable for that comment
    c. More codes apply to the current annotation

    d. The selection of the issue report text snippet for annotation is incorrect
    e. Issue category need to be corrected
    f. Etc.
5. After completing the review, put 1 in the *reviewed?* column in the *issues* sheet.

Note: In case, any annotation cannot be located the reason may be that the annotated comment is hidden on the issue webpage. In these cases, please expand the hidden comment to see the annotation.

11 Validating the Annotations

The last step of the issue annotation process is to validate the coding. Both the coder (annotator) and the reviewer will be involved in this step.

11.1 Coder

1. Traverse the assigned issues again and check the annotations that have comments (made by the reviewer).
2. If the coder agrees with the comment made by the reviewer
    a. Update the annotation
    b. Do not delete the previous annotation. We want to keep all the history.
    c. We can do this: if the coder coded an issue comment as POTENTIAL_SOLUTION_DESIGN, but the reviewer proposed it should be SOLUTION_REVIEW and the coder agrees with the reviewer, then
        i. Change the code (first annotation) to SOLUTION_REVIEW.
        ii. Make a reply with "previously it was POTENTIAL_SOLUTION_DESIGN". ***Note: the first letter is a lowercase letter.***
        iii. If the reviewer proposes to remove any annotation (not needed) and the coder agrees with that, then just rewrite the annotation with lowercase letters.
3. If the coder disagrees with the comment made by the reviewer, then the coder will make a reply to that comment in the annotation.
    a. Both the coder and reviewer will meet to discuss these cases to reach an agreement. Finally, the reviewer will update that.

11.2 Reviewer

1. The reviewer will make a final pass on all the issues after the discussion with the coder, make necessary changes (if needed), and finalize the validation of the annotation.
2. After completing the validation, put 1 in the *validated?* column in the *issues* sheet.