# GitHub Coding Manual

Samridhi Shree Choudhary

November 2017

## 1 Artefact Modification Request and Rebased Request

There are three primary steps to be followed when identifying an *Artefact Modification Request* in an issue thread discussion. The steps should be followed independently for each issue thread. We need both the conversations and the commit activities associated with a Pull Request (PR) or an issue for qualifying this tag.

1. Identify the *Request* for modification.
2. Observe if the *Request* is accepted and is being *Acted Upon* in the current burst of time hence locating the commits associated with the requests made.
3. Catalog the changes associated with the identified commits as *High Potential Change* or a *Low Potential Change*.
4. If there are no explicit commits identifiable for the accepted changes but there is evidence of the change being made in the conversations, mark it as rebased request.

### 1.1 Identify the Request for Modification

The first step requires one to look at the issue thread discussions. Following are the steps to be followed:

- Isolate the *requests*, *suggestions*, *demands* or *revisions* made in the thread of discussions.
- If the above identified requests are explicitly and provably related to either the current PR or any other issue or PR in the burst then proceed forward. The requests that are associated with the current PR should be in reference to the work done in the current burst and should exclude the work done in the PR before the burst. This relation can be found by looking at matching keywords or component names that have been modified in the

burst. There might be an explicit reference to an issue by quoting their issue id. Some discussions might be a continuation of the the discussion in some other PR in the burst.

- If the request made is unrelated to the current or the other burst PRs it becomes a *Feature Request* and is not a modification. We exclude such feature requests for further processing.

#### 1.1.1 Positive Examples

**Example 1 :** In the following example PR, the purpose of the PR (concretizing the behavior of 'ls') can be traced back to the discussions of a different PR in the project burst where people discussing felt a need to have concrete description of the 'ls' functionality of the s3 file system. **PR #3** is markes as as "Request for Modification" in this example.

**PR #3:**

**Person 1:** *Concreteize required outputs of ls, depending on whether bucket exists, and filling in implied directories.*
*e.g.,* ***code***
**Person 2:** *I think so? I don't know that I have enough experience with S3 to be sure. What are your thoughts?*
**Person 1:** :*I will think about this for the evening. I did a little trial and error and am not really happy with the various combinations. There are additional complexities, such as* **code** *and* **code** *both being valid keys with or without data, with or without content at* **code**

...

...

**Related discussion in PR #2**

...

**Person 1:** *Should we raise a 'FileNotFound' error if the user doesn't provide a nice location?*
**Person 2:** *It isn't obvious to me. S3 doesn't actually have directories.*
**Person 1:** ···

...

**Person 2:** *I would appreciate your help, in that case, on making a thorough set of conditions and test for 'ls' - I found it a bit tricky to decide what it should do, is it always the same as glob(path)?*

...

*(more conversation)*

...

**Person 2:** *Would it be fair to say that this is an enhancement unrelated to the PR, so should merge and make a new PR that does this well? PR #1 isn't even in yet.*

**Example 2:** Following is an excerpt from a thread of discussions where a request is made to add some more unit tests for the changes made in the current PR. Hence, the current PR is marked with the presence of "Request for Modification".

**Person 1:** *For the mock data I am a bit unsure how to do that, since we would have to try the same test with multiple versions of data ( one for older EOS releases and one for newer EOS releases) I have added data to cover the on board sensor case.*

**Person 2:** *@Person1 napalm-eos is one of the few napalm drivers (yet) using the new testing framework allowing us to test multiple scenarios, multiple OS versions etc. ⋯ ⋯ ⋯ Is it more obvious now?*

**Person 3:** :*Yes, please, add a new test case. Should be fairly easy to add and I think @Person2 is guiding (ask if you have questions). That will allow us to ensure we do regression tests and that we also test corner cases.*

*Thanks for the fixes, keep them coming ;)*

...

...

### 1.1.2 Negative Example

In the following example, a suggestion is made for adding a logging component for some classes in the project. This suggestion could not be related to any other active PRs or issues in the burst (either by mentioning the classes or the logging functionality in other classes in their thread of discussions). Therefore, we do not mark this PR as a "Request for Modification", instead it is just a "Feature Request". This PR is an example that is discarded for further processing.

**Person 1:** *The Threaded,DerivedArtifact classes already have try-except blocks around their derivation functions to log (and re-raise) exceptions, but other classes should have similar logging:*

- *ExtantArtifact*
- *TransformedArtifact*

...

...

**Person 2:** *I believe this is now closed.*

## 1.2 Observe if the Request is Acted Upon

In the second step, we examine the discussions that have been identified to contain requests for modification. We look at both at the conversations and the commits that are associated with the PR and take the following steps to find if the suggested request was *Acted Upon:*

- If the suggestion made is accepted (as evident through the discussions) and is worked upon in the current burst of time (as evident either through the commits associated with the PR thread or through the verbal confirmations/indicators in the conversation), then we mark it as 'Acted Upon'. In order to find the matching commits we look at the title of the commit and see if it has a similarity or an association with the comments in the thread.

- If the suggestion made is accepted (as evident through the discussions) but is postponed either by opening a separate PR or by talking about doing it in future with no corresponding commits in the current PR, then we do **not** mark it as 'Acted Upon'.

- If the suggestion made is not accepted (no current or future activity), then we do **not** mark it as 'Acted Upon'.

Of all the requests identified in an issue thread by following step 1 described above, even if one of them is accepted and acted upon, we mark that thread as 'Acted Upon'.

Following are some positive and negative examples for identifying if a request is acted upon.

### 1.2.1 Positive Examples

**Example 1:** In the following example, a person submits a pull request summarizing the changes being made. Person 2 reviewing the changes feels there are some issues and lists them. The suggestions made by Person 2 are accepted and the thread is followed by a commit with the title suggesting that it contains the changes suggested by the code review.

> **Person 1:** *Code was written for small screens first, then for the large.*
> *Summary: navbar collapse animation, some styles for sidebar put back, smooth animation while toggle sidebar items, fixed page container*
> *Tested on: Firefox, Opera, Chrome. Need to test on: Safari*
> **Person 2:** *Can you make sure it is up to date with dev please?*
> **Person 1:** *Yes, the code was updated with the current dev.*
> **Person 2:** *There are still few issues.*
> *The sidebar initial state is open and than it collapses, you can see that by refreshing the page and see it collapsing.*

*The navbar when minimised has two issues:*

...

*You are still using styling both in the main sidebar css function and in the skin function.*

...

**Person 1:** *Thanks for review, I'll do the fixes soon.*

...

(commit followed with title:"fixes sidebar css")

**Example 2:** Following is the same as example 2 in section 1.1.1. The suggestion made to add new tests is accepted and is associated with a commit in the same burst with a corresponding title.

**Person 1:** *For the mock data I am a bit unsure how to do that, since we would have to try the same test with multiple versions of data ( one for older EOS releases and one for newer EOS releases) I have added data to cover the on board sensor case.*
**Person 2:** *@Person1 napalm-eos is one of the few napalm drivers (yet) using the new testing framework allowing us to test multiple scenarios, multiple OS versions etc. ········· Is it more obvious now?*
**Person 3:** :*Yes, please, add a new test case. Should be fairly easy to add and I think @Person2 is guiding (ask if you have questions). That will allow us to ensure we do regression tests and that we also test corner cases.*

*Thanks for the fixes, keep them coming ;)*

...

(commit followed with title: "Tests for newer versions of EOS")

### 1.2.2 Negative Examples

**Example 1:** In the following example, the suggestion made is postponed as visible through the following discussions:

**Person 1:** *Should we be consistent with use of % or {} ?*
**Person 2:** *Yes I think it deserves a separate PR*

**Example 2:** The following example is the same as the example 1 in section 1.1.1, where with respect to PR #2, the suggestion made is postponed by opening a separate PR thread.

...

**Person 1:** *Should we raise a 'FileNotFound' error if the user doesn't provide a nice location?*
**Person 2:** *It isn't obvious to me. S3 doesn't actually have directories.*

> ...
> *(more conversation)*
> ...
>
> **Person 2:** *Would it be fair to say that this is an enhancement unrelated to the PR, so should merge and make a new PR that does this well? PR #1 isn't even in yet.*

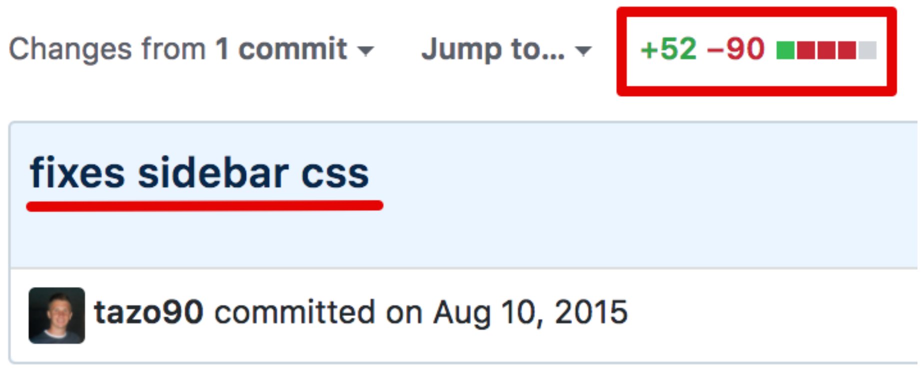
## 1.3 Identify High and Low Potential Changes

In this last step, the filtered *Acted Upon* requests are processed to extract the commits associated with them. These commits represent the associated change. The changes that have high impact are likely to take more time than the ones with low impact. In order to make this distinction, we look at the *number of lines of code changed*. Even if one of the commits has **greater than 30 lines** of change, we mark them as **High Potential**. Anything **less than 30 lines** of change is marked as **Low Potential**. The number of lines of code changed is calculated as the **sum** of the lines added and deleted.

Therefore, the requests that are accepted, acted upon and are associated with high potential commits are marked with this tag.

### 1.3.1 Positive Examples

**Example 1:** The following example is the commit associated example 1 in section 1.2.1. The commit titled 'fixes sidebar css' has more than 30 lines of code change as seen below:

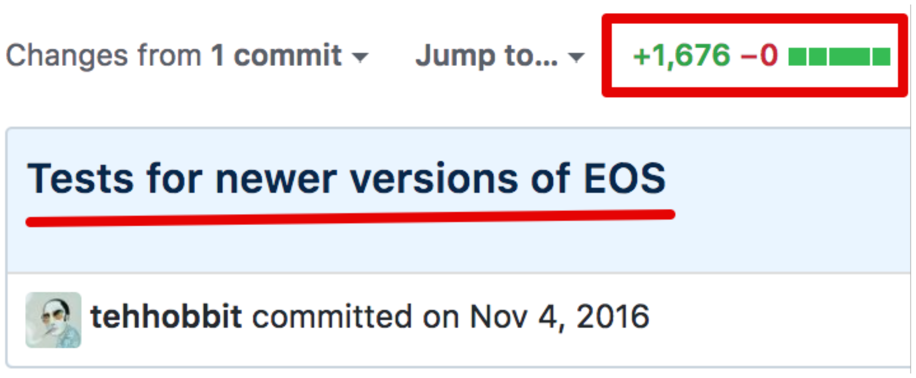


fixes sidebar css

tazo90 committed on Aug 10, 2015

Figure 1: Number of lines of code changed for the commit

**Example 2:** The following example is the commit associated example 2 in section 1.2.1. The commit titled 'Tests for newer versions of EOS' has more than 30 lines of code change as seen below:



Changes from **1 commit** ▾ **Jump to...** ▾ +1,676 −0

**Tests for newer versions of EOS**

**tehhobbit** committed on Nov 4, 2016

### 1.3.2 Negative Examples

**Example 1:** The following example shows the excerpt of the PR thread where a modification was suggested and was acted upon with a corresponding commit. However, the commit has less than 30 lines of code change.

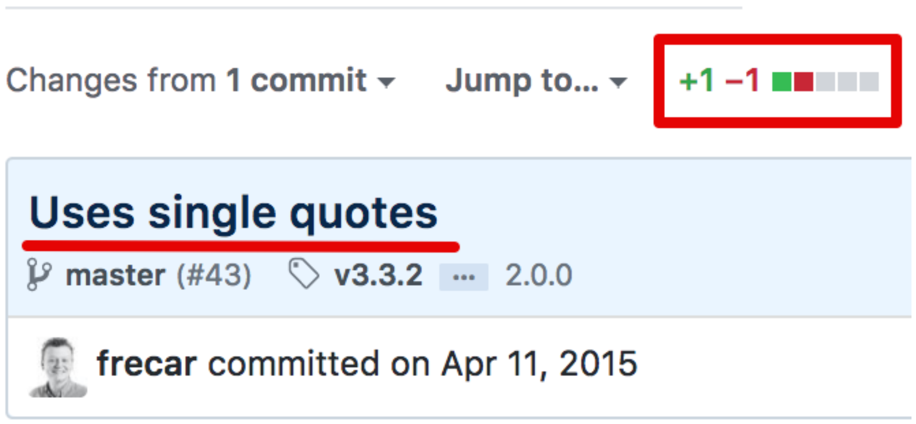
. . .

**Person 1:** *Please use ' instead of " for consistency.*

. . .

. . .

(commit followed with title:"Uses single quotes")



Changes from **1 commit** ▾ **Jump to...** ▾ +1 −1

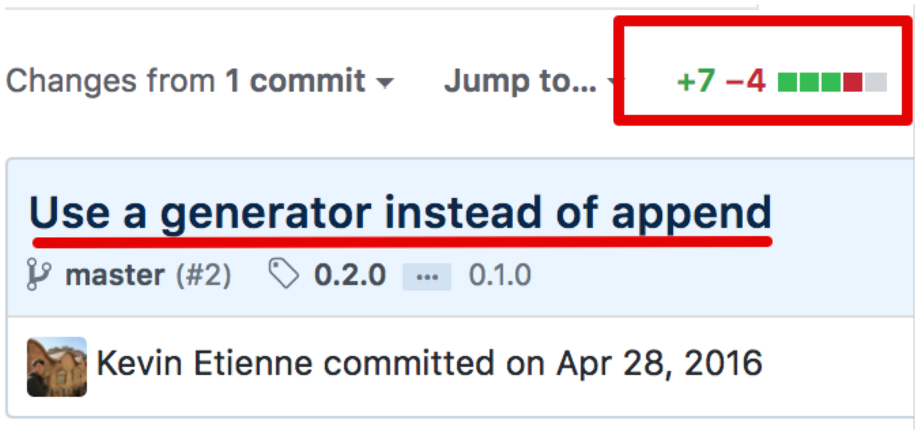
**Uses single quotes**

**master** (#43) **v3.3.2** ... 2.0.0

**frecar** committed on Apr 11, 2015

**Example 2:** The following is a similar example showing accepted suggestion with a corresponding commit.

> ...
>
> **Person 1:** *I'd be tempted to use a 'generator' instead of 'messages' and 'append'.*
>
> ...
>
> (commit followed with title: "Use a generator instead of append")


Changes from 1 commit
Jump to...
+7 −4
Use a generator instead of append
master (#2) 0.2.0 ... 0.1.0
Kevin Etienne committed on Apr 28, 2016

## 1.4 Rebased Requests

This is a separate category spinned off from the *Artefact Modification Request*. There might be PR threads on github where the modifications were suggested and the changes were made, however the commits associated with all those changes were rebased and combined into one single commit before merging the PR into the branch. In these situations, it is intractable to identify and separate out the commits associated with each individual request. We mark these PRs with the tag *Rebased Request*. They follow the same flowchart as shown in figure 2 up until the *acceptance of the requests*. Following are some examples of this category

### 1.4.1 Examples

**Example 1:** In the following example, the suggestion made is accepted as visible through the verbal confirmation on the issue thread conversation but there is only one large commit associated to the entire PR thread and not the specific change as such. We do not mark these examples with this specific category but rather as *Rebased Request* :

> ...
>
> **Person 1:** *If we had a more general* **code** *class, you could do something along:* **code**. *For example. Not saying that a* **code** *is*

*a no go, but building it from* ***code*** *maybe funnier ;-)*

. . .

**Person 2:** *@Person1 something like this '?*

. . .

*(more conversation)*

. . .

**Person 1:** *@Person2 This looks perfect. Can you fix the pep8 error and rebase/squash your branch into a proper clean commit? (or let me know I'll do it)*

. . .

Figure 2 shows the final flowchart for identifying both *Artefact Modification Request* and *Rebased Request* in an issue thread discussion.

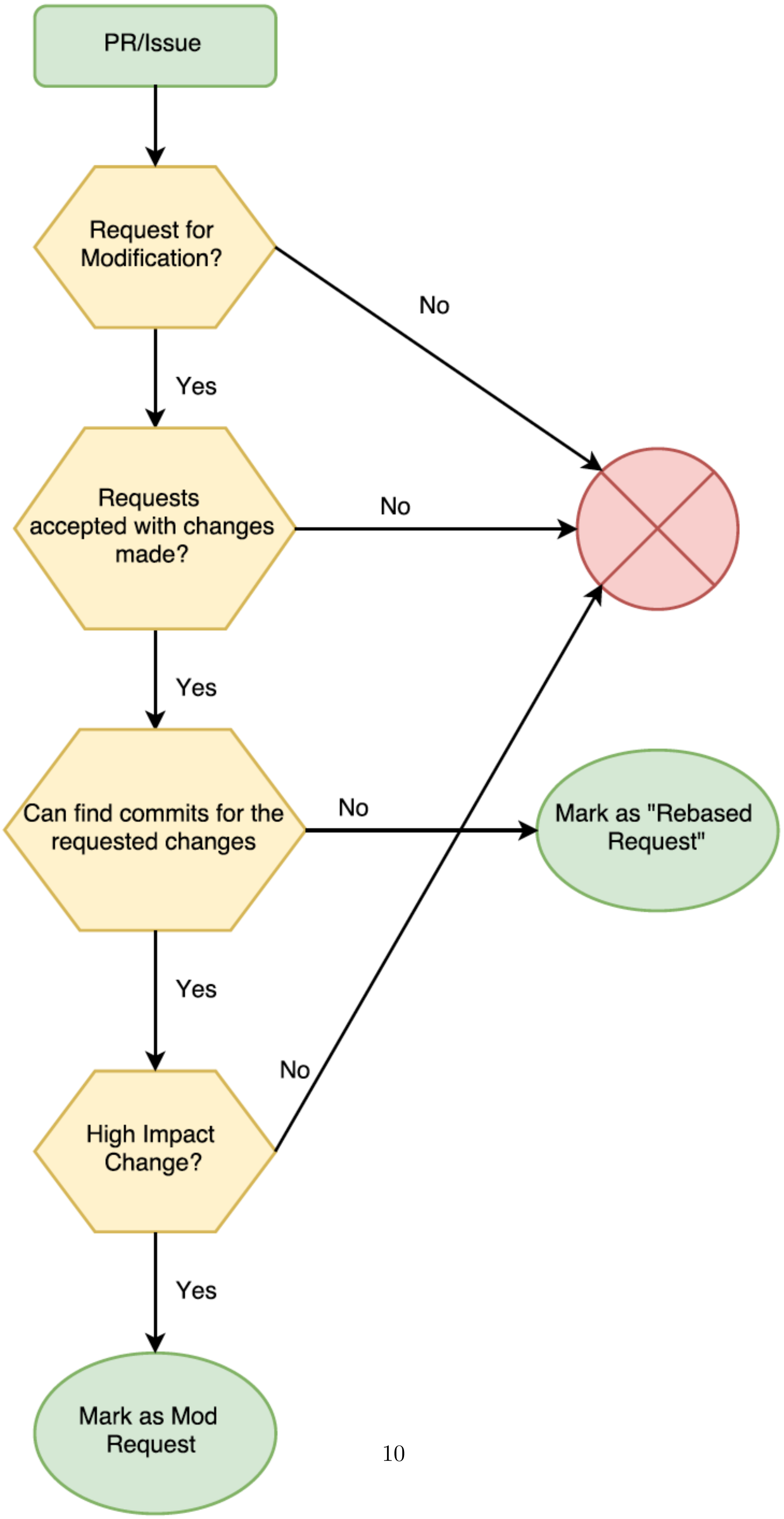

PR/Issue
Request for Modification?
No
Yes
Requests accepted with changes made?
No
Yes
Can find commits for the requested changes
No
Mark as "Rebased Request"
Yes
No
High Impact Change?
Yes
Mark as Mod Request

Figure 2: Flowchart for identifying Artefact Modification Request and Rebased Request

## 2 Testing

Figure 3, outlines the actions to be followed in order to identify testing in an issue/PR thread. For any given PR to be qualified with this tag, it needs to show evidence of the tests actually being run. These evidences should be explicitly present in the conversation text of the threads.

Following points should be kept in mind while marking for this tag:

1. The conversations should mention the execution of tests in the bursts.
2. The automated Continuous Integration tool messages should be ignored. The examples of CI tools in GitHub for the samples investigated so far are *Travis*, *Coveralls* and *codecov-io*.
3. However, if the conversations discuss either *Travis* or *Coveralls* regarding their execution, they should be considered.
4. Ignore the messages that are related to the reproduction of the issues filed.

### Positive Examples

**Example 1:** In the following example, Person 1 submits a PR. However, Person 2 explicitly says that he will be testing before the merge happens.

...

**Person 1:** *Various small fixes. Look at the commit history for more details. Keep in mind that we will move soon to a more robust way for logging configuration.*
**Person 2:** *At a first glance, everything seems fine. Could you explain the "pin security" thing?*

...

*(more conversation)*

...

**Person 2:** *I will test NOW*

**Example 2:** In the following example the people involved specifically discuss about Travis running the test and hence is considered as a positive example for this tag.

**Person 1:** *Let's see if Travis build this :)*
**Person 2:** *Take that for lunch, Travis!*

...

**codecov-io:** *..Automated message..*

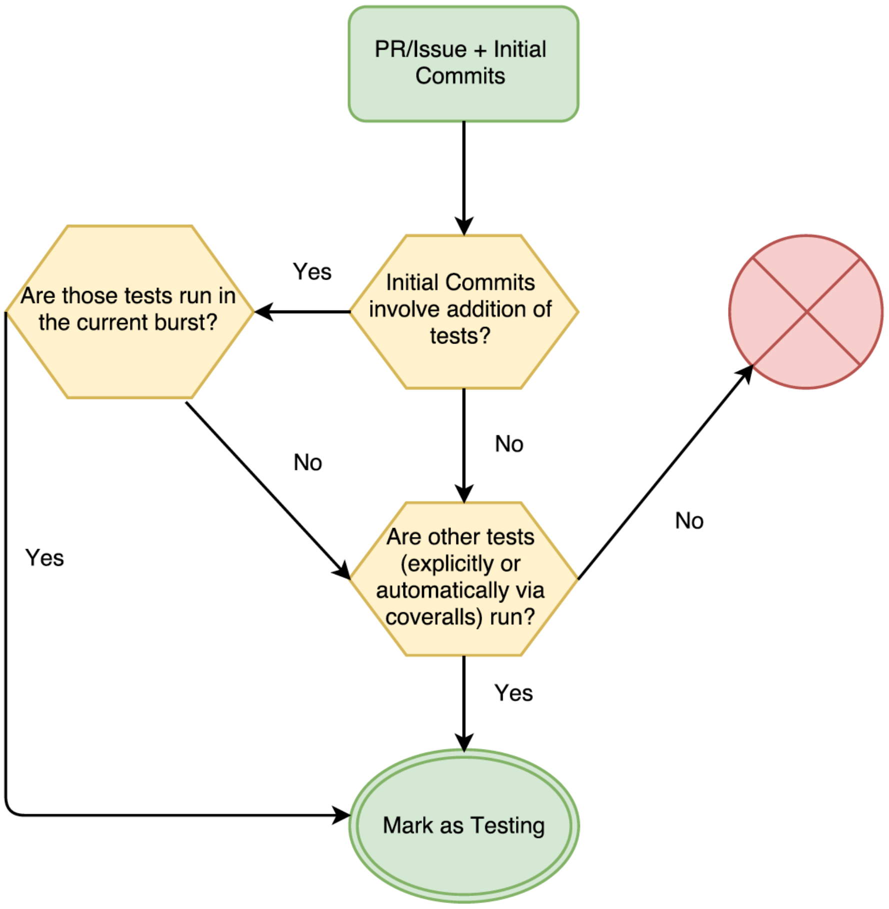

PR/Issue + Initial Commits
Initial Commits involve addition of tests?
Yes
Are those tests run in the current burst?
No
No
Yes
Are other tests (explicitly or automatically via coveralls) run?
No
Yes
Mark as Testing

Figure 3: Flowchart for identifying Testing

__**Example 3:**__ In the following example Person 1 and 2 discuss of different ways to test the PR that contains the changes related to json decoding for new Flask Version. This is concluded with Person 3 actually testing the changes explicitly.

> **Person 1:** *Preparing json deconding for new Flask version (1.0)*
>
> ...
>
> **Person 2:** *Ok, this is blocking, but is very difficult to test. There is no unit testing, and the API is not documented. What should I do?*
>
> ...
>
> *(more conversation)*
>
> ...
>
> **Person 3:** *I tested this, I will merge tomorrow if you dont have further comments*

**Example 4:** Following is a similar example where an explicit testing of the changes made can be identified from the conversation thread.

> **Person 1:** *Page to modify metadata of volumes. Finally! This is UI stuff, thus it is difficult to make systematic tests. I hope all of you will test this new interface. Requires #247.*
> **Person 2:** *I see that you are just requiring authentication to edit a volume. Shouldn't it be based on authorization, instead of authentication?*
> ...
> *(more conversation)*
> ...
>
> **Person 3:** *I tested it (w/o users) on flask 0.10.1, it works smoothly!*
> **Person 3:** *@Person3 welcome aboard and thanks for testing.*

## Negative Examples

**Example 1:** In the following example suggestions are made to expand the exisiting set of tests to cover the new changes introduced in the PR. The PR is merged with the new tests being added but not being explicitly run in this time interval.

> **Person 1:** *Tenant.auth_user may be NULL. Fixes SENTRY1Z9 @Person2 @Person3 I'm not sure if whatever handles this response will account for this, but the column is definitely nullable.*
> **Person 2:** *fyi i think we have tests for this endpoint so could probably extend it to have one without auth user*
> ...
> *3 commits related to adding the tests as suggested above.*
> ...
>
> **Person 3:** *Added test and added fix for the UI to handle this.*
> *PR Merged*

**Example 2:** Following is a similar example. The objective of the PR is to add more unit tests. However, they are never run and the PR is merged immediately after the commits.

> **Person 1:** *Added more unit tests for util.py*
> ...
> *PR Merged*

## 3 Design Discussion

Developers often discuss feature additions, bug fixes, code contributions or reviews in the issue threads by posting “issue comments” or “pull request commit comments”. These discussions might either be related to changes that are trivial or the changes that are complex. If there is no right way doing a task, people often end up discussing trade-offs of different approaches. We want to identify these discussions where people collectively discuss and decide on the *specifications* and the *functionality* of the project. Figure 4 shows the actions to be followed in order to identify this tag. Following are some key points to keep in mind during this identification:

1. There should be atleast two people involved in the discussion.
2. We should look only at the discussion and not the associated commits or the changes if present.
3. We should look only at the discussions that happened within the boundary of the burst.
4. Discussion should be about functionality and specifications of the components and not the routine or conventional work practices.
5. However, if the work practices being discussed are not routine or are in the flux of change where people are discussing the right way to do it, we should consider that discussion.

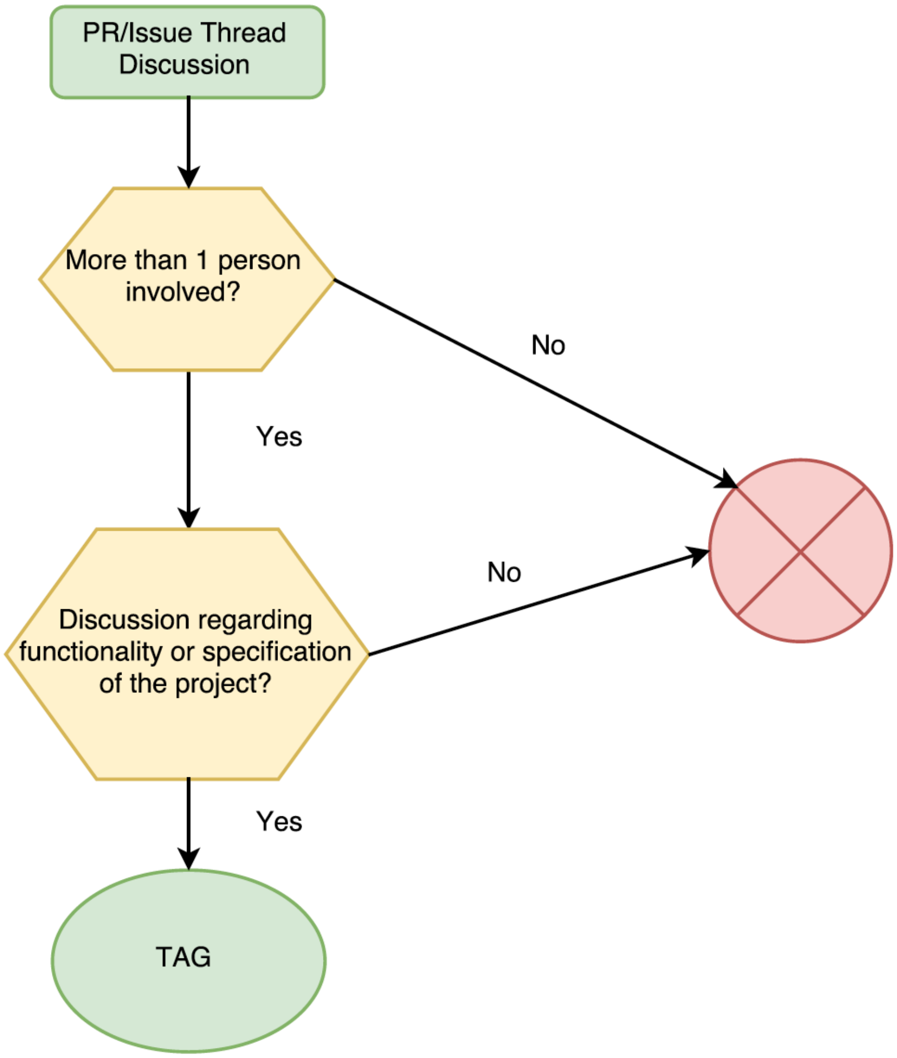

PR/Issue Thread Discussion
More than 1 person involved?
No
Yes
Discussion regarding functionality or specification of the project?
No
Yes
TAG

Figure 4: Flowchart for identifying Design Discussion

## 3.1 Positive Examples

**Example 1:** In the following example people discuss about the behavior of 'ls' for their file system. Each of them have their own concerns regarding the functionality which they discuss.

> **Person 1:** *Concreteize required outputs of ls, depending on whether bucket exists, and filling in implied directories.*
> *e.g.,* ***code***
> **Person 2:** *I think so? I don't know that I have enough experience with S3 to be sure. What are your thoughts?*
> **Person 1:** *I will think about this for the evening. I did a little trial and error and am not really happy with the various combinations. There are additional complexities, such as* **code** *and* **code** *both being valid keys with or without data, with or without content at* **code**
> **Person 1:** *Let me step back and describe a simpler ls that more closely reflects the workings of S3 (which has no real directories, only paths that happen to contain / ). What if we go back to a flat structure, allowing arbitrary filenames?*
> ...

**Example 2:** In the following example people discuss about the UX specifications of an app and discuss the trade-offs for the design decision made about a UI element.

> ...
> **Person 1:** *Now a new user has to type over someone else's name (old user). It's very inconvenient in my opinion. What are proper solutions for that?*
> **Person 2:** *What do you mean, "has to type over someone's name"? They can press Backspace or Delete key if they are as incompetent as my grandparents, otherwise on any first key press, the field will clear, replacing the contents with that first letter.*
> **Person 1:** *It doesn't clear. The text is only highlighted, which means pressing Backspace again to get empty field. I don't feel good having other people's name there, so demonstrators will have to delete this over and over again. I'm just trying to make it easy with a click of a button.*
> **Person 1:** *Yes, or you can just focus the field and start typing the new name (saves one keypress). There are no privacy concerns. Orange is a single-user app.*
> ...
> ...
> **Person 1:** *Done.*

## 3.2 Negative Examples

**Example 1:** In the following example Person 1 and Person 2 discuss about changing a variable name in the code. It does not specifically relate to changing the functionality or the specification of the API and therefore should not be considered as design discussion.

> **Person 1:** *This isn't 'last_login', it's the notification boundary date-time.*
> **Person 2:** *Do you have a suggestion I'm not sure what to replace it with?*
> **Person 1:** *'boundary'? 'notification_boundary'? 'inactive_boundary'? It's not a very easy one to name I agree.*
> **Person 2:** *Cool cheers*
> **Person 1:** *'threshold' is another potential term to work with.*

**Example 2:** In the following example Person 1 and Person 2 discuss about practices followed by the people in the project regarding opening new PRs and organisation of test cases. The discussion does not explicitly relate to a functionality change. There seems to be a right way to do things that is conveyed through the discussion which is therefore not regarded as a design discussion.

> **Person 1:** *Do you guys want this as one large issue or as several issues ?*
> **Person 2:** *Hi Person 1,*
> *It's fine fixing them in one single PR. But it would be great to open an issue per bug. Doing so, you can then create a new test case per issue providing the specific mock data, as you see on your device (what triggered the bug in your case). For example: ⋯ ⋯. Additionally (but not mandatory), if you have new lines, you can also add a short inline comment in your code to point the issue ID.*
> *Does this make sense to you?*
> ...
> ...
> **Person 1:** *For the mock data I am a bit unsure how to do that, since we would have to try the same test with multiple versions of data ( one for older EOS releases and one for newer EOS releases) I have added data to cover the on board sensor case.*
> **Person 2:** *@Person 1 napalm-eos is one of the few napalm drivers (yet) using the new testing framework allowing us to test multiple scenarios, multiple OS versions etc.*
> *Let's look again at the example #58 ⋯ ⋯. Is this more obvious now?*
> **Person 2:** *Thx for the last commit, now we are talking. Shouldn't you revert the changes to the "normal" test case, though?*
> ...

## 4 Semi Automated Coding Process

This section is a reference guide for the new set of files obtained by automatically extracting information related to the tags and arranging them in a way that is easily readable and comprehensible.

### 4.1 Artefact Modification Request

**File Format**: The entries in this file are arranged as follows:

- The file is divided into sections with each section representing a single commit and consecutive sections separated by two blank lines.
- The commits displayed are filtered based on their number of lines of code change and their time of commit. Only the ones that have greater than 30 lines of change and were committed after the pull request started are used for this file.
- Each section starts with the commit message associated with the commit followed by the comments in the pull request thread that have maximum word overlap with the message and occurred before the commit itself.

**Coding Process**: For each section in the file, the commit message (first entry in the section) is matched to the comments in that section to find a comment that requests the change represented by a commit. Following are a few caeats to take care of before marking this as a modification request:

- If the commit message itself indicates that it is fixing suggestions or incorporating reviews, it should be marked as modification request.
- The matching comment identified should be made by a person different from the person committing the file.
- If the comment is a quoted text, care should be taken to see that the quoted text that makes the suggestions should be made by a different person and not the person committing.

If any such commit in the file is identified to be a modification request, mark the burst as having a modification request.

### 4.2 Testing

**File Format**: The entries in the file contain filtered comments that contain the word 'test' in all its inflected formats and the automated CI tools (Travis, Coveralls, codecov-io) arranged chronologically. These comments are filtered to fall within the burst boundaries.

**Coding Process**: The entries should be checked to see that these comments represent conversations that show the execution of the tests and not just the traces of a test file. If any evidence of the execution of test is found, mark the burst as having testing.

## 4.3 Discussion

**File Format**: The entries in this file are arranged as follows:

- The file is divided into sections, with each section containing a thread of issue comments and the consecutive sections separated by two blank lines.
- The comments in the thread are arranged chronologically and are filtered to contain atleast 2 different people participating and to fall within the burst boundaries.

**Coding Process**: Each section should be investigated independently to see the traces of design discussion following the steps as mentioned in the section 3. If any such thread is identified as design discussion mark the burst as having design discussion.