

# Generalized Review System

Nithesh N. Hariharan

## Why are you interested in working with Oppia?


I found out about Oppia in November and started working with the team since then. Among other things, there are two things that I really liked about Oppia. Firstly I like the goal of the organization. I have used a lot of online courses and play along courses, but none of them were as innovative and user friendly as the Oppia platform. This platform is simple enough to use so that it can be given to a student of primary school and the student will be able to play through the lessons without any issues. This goes to say how much impact this project could create in the society. It would help students in various parts of the world, where the education system may not be up to the mark. Also by taking a non conventional method to teaching than the traditional classroom approach, it would help students to develop a different mindset to learning.

I was also involved in discussions about the recent RCT that happened in January. The feedback we obtained got me thinking. Each student is different and their grasping powers are varied. In order to understand a concept, the student needs to be motivated to think about it. It is also equally important to catch their attention for that span of time. This is rarely what happens in school. With a platform like Oppia, the learners can take a lesson at their own pace, keep trying until they get the solution right, and also get hints to guide them through their path to learning. This system is very revolutionary and could help supplement the primary education system throughout the world. I would like to be a part of such an organisation!

Secondly, the community is just awesome! While working on various issues, testing releases, and in general, meeting with the community, I have spoken to a lot of members on the team. All of them are very helpful and always ready to guide you. It had been a great learning experience for me so far with Oppia. I hope that I can keep contributing and learn a lot more in the process.

## What interests you about this project? Why is it worth doing?

Oppia is an organisation that relies on the community not only for developing the infrastructure but also to get good quality content on the site. The explorations that are present on the website are all created by the community for the use of the community. The key explorations in Oppia are currently created by a smaller set of people mostly a part of the development team. We would like to allow people from the community to also help out to create and contribute to key explorations.



But just allowing more number of contributors may not be the best idea. Too many cooks spoil the broth. To allow valuable contributions by other members of the community while maintaining quality of content produced, we can have a system where the contributions of the community are put forth to the creators, and they review the changes. If they are happy with the changes, they should be able to accept them into the exploration. This allows us to effectively crowdsource Oppia's content while maintaining quality.

I feel that adding this system is really important for the organisation. It allows us to increase our contributor count while also maintaining quality. This will enable us to produce many more better explorations, at a much faster speed.

## Prior experience (especially with regards to technical skills that are needed for the project)

I have worked on multiple projects involving web development. I have worked as a backend developer for an educational startup [Ylurn](#), I have also worked on a platform for my school, to help teachers conduct weekly assignments and tests on an online platform, and the answers are evaluated immediately. I have worked on some other hobby development projects where I learnt how to use web development technologies.

I have been contributing to Oppia since November 2017. I have helped to fix some issues from the issue tracker and also helped in the release testing for the releases starting January 2018. In the process I have acquainted myself with the structure of the codebase, the workflow and also the release process.

I have actively participated in programming competitions (including Google Code Jam, ACM ICPC and Codechef Snackdown). I have also actively participated in weekly and monthly contests hosted on various online websites. This has enhanced my ability to think about efficient solutions for problems, and also consider various corner cases while implementing. Debugging code is another skill that I have improved on while competing in such events.

## Links to PRs to Oppia

- Fractions landing pages: [#4758](#), [#4801](#)
- Additions to Fractions interaction: [#4387](#), [#4400](#), [#4645](#), [#4648](#)
- Many more PRs for blocking bugs, issues, upgrading libraries etc, For a full list please click [here](#)

## Overview

This project aims to introduce a system to introduce a suggestion-review system which would allow the community to contribute to the content on Oppia. It also aims to add a review system to allow easy management of the proposed suggestions and accept or reject them as appropriate. In order to maintain quality of reviews, we allow reviews from users who have made above a certain number of contributions in similar kind of suggestions. For each exploration the creator of the exploration can appoint some trusted reviewers who will be able to accept (merge) content related suggestions for that exploration. At a site-wide level, translation reviewers can be selected for each language and they can accept translations for that particular language.


## What is an apt name for the framework: *tasks* or *suggestions*?

The dictionary defines a task as “a piece of work to be done or undertaken”. Whereas a suggestion is defined as “an idea or plan put forward for consideration”. As the proposed changes would be reviewed before they are accepted, the definition of a suggestion seems more apt for the context. So we go ahead by calling it the “suggestions framework”. Possible suggestions include changing or adding content to the website, supplementing the existing content with translations, images, etc. For some advanced use cases (a possibility), the options that you get in the admin page (to add new roles, changing the email properties, etc) suggestions can be created and put forth for review.

## Why do we need suggestions?

By making all “ideas put forward for consideration” on Oppia pass through another layer called suggestions, we have room for more validation. As all of these suggestions will have a similar base structure, it will allow us to build a general system of validation, which we term as the generalized review system. For any of the suggestions created, it would be best to have a second pair of eyes look at and validate what you have done. This will help ensure quality of products delivered to the learners and also help us widen the user pool who can actually make changes to content.

Teachers always mention that learning is never brought about effectively by having one teacher teaching and students getting taught. It is a mutual learning experience. The suggestions of students are valuable and will help better the educational content on Oppia. Their suggestions need to be presented to the teachers for review. There may also be



situations where the learner creates a suggestion to make slight wording changes in the content, or to report that the translations do not match the content.

As we keep adding more and more features to Oppia, it is important that we have a framework where both the creator and learner sides of the community can work together to provide better lessons. Also this system should easily be extendable to newly added features.

## Types of suggestions

A general system should be defined in order to manage various different types of suggestions that could arise. A list of few of the suggestions that can be added are:

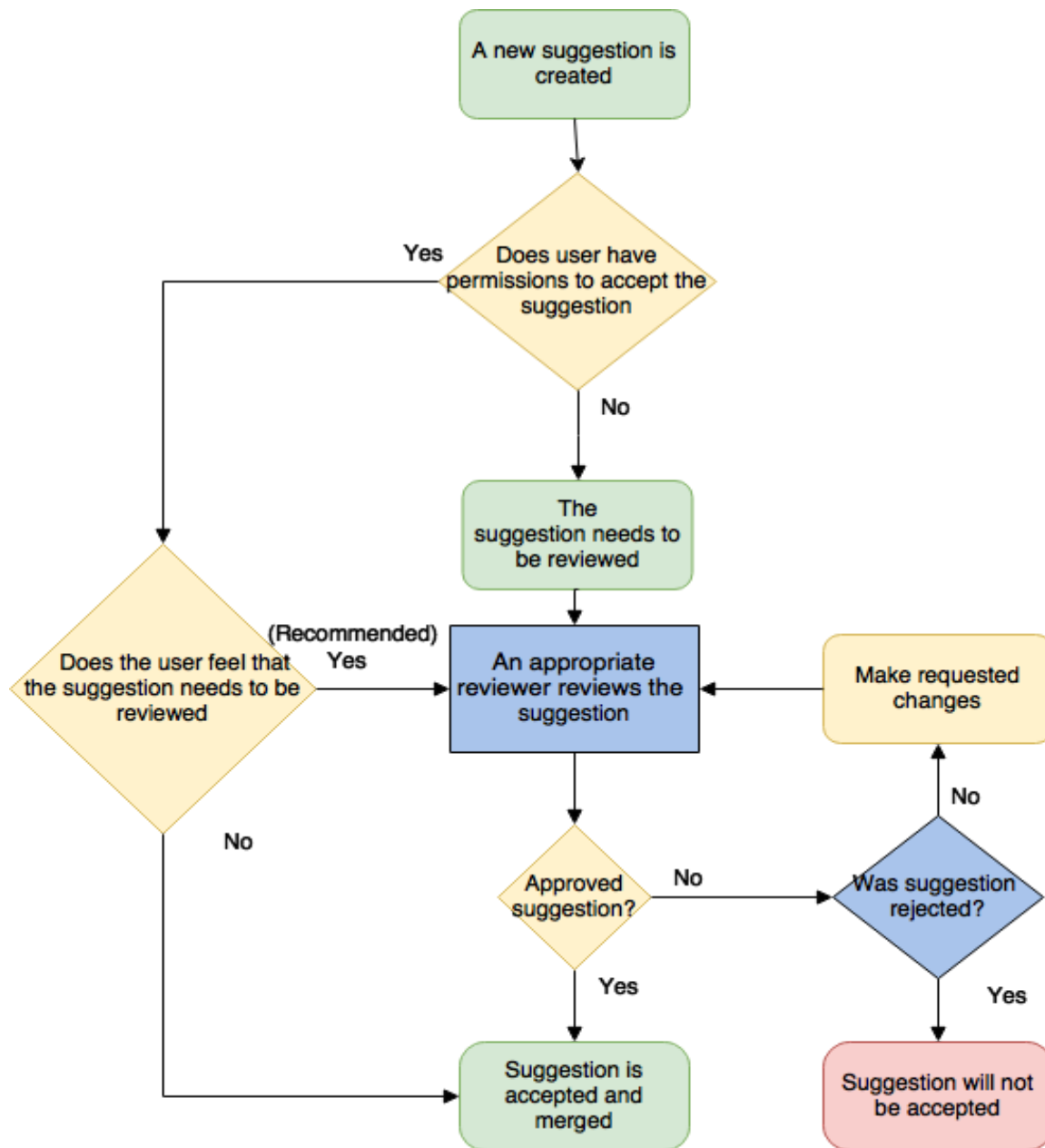
- Edit the content of an exploration state, including the concept card, answer groups, hints, solutions, etc.
- Add or change translation files for some content of the exploration.
- Adding demonstrative images for a lesson.
- Adding a question.
- Edit title, category, tags and other collection properties.
- Make edits to properties of stories and topics (Features that will be added soon to Oppia!).

These suggestions can be broadly classified into the following:

- Content related suggestions which relate to making edits for the components of a lesson, the lesson flow, etc.
- Suggestions that create new entities (questions, explorations, collections, etc)
- Suggestions relating to audio (translation) uploads and image uploads.

## Technical outline

### User control flow upon creating a suggestion



## Overview of new features

The above flow chart describes the general flow of control during the process of creating a suggestion and getting it reviewed. A feedback thread for a suggestion would be necessary where the proposed edit or addition can be discussed about and reviewed before being accepted. For some situations it can be allowed (though not recommended) to directly accept the suggestion without a review. This can be used by users with edit access for the activity or by trusted translators. Bypassing the review system should be used as a last resort only when running close to deadlines.

A basic scoring system will be implemented for users to keep track of number of contributions (Each contribution is defined as an accepted suggestion). A good metric would be to store the number of contributions in a particular language and number of contributions for a particular category of lessons. Having more than 10 (say) contributions in a particular category will allow you to review other suggestions within that category. After making 10 audio translation contributions for a particular language, you will be allowed to review translations in the language across explorations. This will allow us to maintain quality of reviews and translation content on Oppia.

Every suggestion, in general, should go through the review process. If it is created by a user with edit access, the user is given an option to get the suggestion reviewed. Typically we would like the user to get the suggestion reviewed before it can be accepted, whereas if the user needs to get the suggestion in immediately (say, just before an important deadline), then the user could bypass the review system and complete the suggestion (provided he has edit access).

Users with score above 10 for content related contributions under that subject category can review the suggested edit. Once the reviewer is happy with the changes, they approve the changes. Then a trusted reviewer (a user with edit access for the activity) can accept (merge) the changes. A new role for *reviewers* will be added. Also a new category of users will be set up at the activity level called *trusted reviewers*. Users with a trusted reviewer role and who are present in the category of trusted reviewers within that activity are termed as trusted reviewers for that activity.

## UX details

This section defines how the user experience would be while creating and reviewing suggestions. It discusses various possible scenarios of creating a suggestion.

### Suggestion 1: Make an edit to the content card of a state

This suggestion is currently handled using the suggestions framework already present. No changes to the current UX will be made while migrating this suggestion to use the newly added suggestions framework. The UX is as follows

1. In the player view, there is a suggest changes button on the navigation bar (At the top right of the page). A learner uses this button to suggest changes.
2. Upon clicking it, a modal pops up with an RTE showing the contents of the concept card of the state the learner was in. The changes are made in the RTE and a button to suggest changes will be presented to the learner. Upon clicking this, the suggestion is created and is sent for review.
3. These suggestions are all viewable under the feedback tab of the creator view. A user with permissions to view the suggestion threads can state approval, or reject the suggestion.
4. For some situations, the reviewer can start a conversation with the suggester using a feedback thread linked with the suggestions. This can be to clarify some point, request for some changes, or any other discussion.
5. Before accepting the changes, a validity check is run on the suggestion. For suggestions of this type, the check would validate presence of the state that is being edited. Between the version where the suggestion was created and the current version of the exploration, the state that is linked to this suggestion should not have been renamed or deleted.
6. A trusted reviewer for that exploration can accept the changes into the exploration. The trusted reviewer can merge an approved suggestion, or state their approval and merge the suggestion.

### Suggestion 2: Make an edit to the answer groups of a state

This suggestion is similar to the first suggestion as it links to an exploration and is also a content related suggestion. However there is no frontend view where the user can create such a suggestion. The UX for reviewing will be the same, the creation of this suggestion (or any other suggestions to) can be done in one of two ways:

#### Approach 1:

1. In the player view, the suggest edit button, on being clicked first shows up a modal with all the components that the learner has viewed, including hints, solutions, feedback from Oppia for the submitted answers and the concept card.



2. The learner chooses one of these components, and then an RTE is shown with the content of that component and the learner can suggest changes as in the previous suggestion.
3. Then the suggestion is reviewed by following steps 3-6 of the first suggestion. The validity check would be similar to the last case. The state where the answer group was edited should still be present in the current exploration version.

#### Approach 2:

1. In the player view, upon clicking the suggest edit button, the learner is redirected to the /create/ page for the exploration.
2. Here beside all editable components we show them a pencil symbol (similar to the when the creator sees the page), but these buttons will just allow suggesting edits to the content and not to edit the content directly.
3. Upon clicking a button, the learner will be shown a modal which will resemble the suggest change modal that is shown in step 2 of suggestion 1. Here the content in the RTE will be the element the learner chose to edit. On clicking a make suggestion button, the suggestion will be created and sent for review.
4. Then the suggestion is reviewed by following steps 3-6 of the first suggestion.


Both approaches I feel have their pros and cons. Approach 2 might be too complicated for a learner to comprehend (with the graph on the side, and various rules and sections). Whereas for some users who are testing the exploration, it would be better for them to have a look at the subtleties and then make suggestions. Approach 1 may result in a messy UI (As we will be trying to crunch down majority of the data from the create page into a modal). This could also result in bad UX.

As a part of this project, the backend changes to facilitate such a system will be built. The UI will be developed after the three proposed milestones.

#### Suggestion 3: Add a new question

This suggestion is different from the previous two suggestions. It involves a different model, and the suggestion is to create a new instance of this model. The UX for this would be quite different from the other exploration related suggestions mentioned above.

To add a new question, the relevant suggestion must be created from a question frontend (Not yet present but will eventually be added). The user will need to input all the required parameters like question name, language code, and the question data. Once the user submits this data, a suggestion will be created to add a question, and will be submitted for review.



This being a feature that is still in development, the best people to review a newly added question at the moment is probably admins. A different interface will need to be added to display all the suggestions to the admins and then the suggestions can be reviewed. Once the suggestion is approved, a new question will be added. The validity check would be to validate that the suggestion is indeed a valid question. This can be done by calling the validate function on the domain object of the question.

**NOTE:** One subtlety in this task is that the feedback threads are linked to explorations. So adding a discussion thread to these type of suggestions would not be possible. The process to allow feedback threads has been outlined in the later part of this proposal, but it will be completed as a future milestone.

#### Suggestion 4: Add or edit audio translations

This suggestion will be linked to any text displayed to the learner which will need audio translations. This UX for this project can be described effectively from the translations dashboard view. A translator adds or edits translations for many parts of the exploration and then submits them. The translations will be sent for review.

Another tab can be added in the translations tab itself to view all the suggested translations. When a reviewer opens the tab, all translations suggested in the language the reviewer is proficient in will be shown. The proficiency in a language is determined by the reviewer's score in that language. The reviewer will have an interface where the reviewer can play the newly recorded translation while having a view that shows the text that was translated. A validity check will be run on the suggestion before merging. This will check that the content that was translated hasn't changed since the version where the suggestion created. If the text in the current version matches the text in the version where the suggestion was created, the suggestion is valid and can be merged.

## Considered implementation method

Add a separate model to handle “new” suggestions and remove the existing “old” suggestions model

Firstly, we add a new model to store suggestions. A suggestion would optionally include a feedback thread to handle review comments.

Once suggestions are implemented, a job can be run to convert all existing suggestion objects to “new” suggestion objects. Once this migration is completed, the “old” suggestions framework is completely replaced by the newly implemented suggestions framework and can be safely removed.

One issue is the fact that feedback thread is still linked to an exploration (necessarily). To allow review comments in any general suggestion, this must be changed. There are two possibilities I came up with:

**Possibility 1:** A feedback thread can be linked to a suggestion instead. This leads us to think about how the rating and feedback system present in explorations would need to be changed. For this we could create another type of suggestions, say “feedback from learners”. This type of suggestion would just convey the feedback message and suggestion. It *cannot* be reviewed or merged.

Intuitively this approach doesn’t fit in as the learner did not actually suggest anything if the feedback message was “Nice lesson” or similar. But it is a possible implementation strategy.

**Possibility 2:** The feedback thread continues to stay linked to an exploration (can be made to link to any lesson instead). The `exploration_id` parameter in the feedback thread model can be made optional, and an extra `suggestion_id` parameter (again optional) can be added. When an exploration ID is specified, the feedback thread will be linked to an exploration (for learner feedback) and when a suggestion ID is specified, the feedback thread will be linked to a suggestion instead. This will allow retaining of the current functionality and also allow it to be linked to a suggestion.

Feedback threads linked to suggestions will not be linked to explorations and vice versa. A suggestion may be made to some exploration content, and may have a feedback thread attached to it. In this case, the thread will be linked to the suggestion and not the exploration.

This possibility seems intuitively better as the feedback thread is generalized to provide feedback for suggestions as well as for lessons. As per this modelling, The existing functionality for user feedback will be unaltered, and functionality will be extended to start a discussion on any suggestion put forward for review.

## Technical details of the backend

### Suggestions

The first model to be added into the backend is for a suggestion.

Several constants need to be defined for some parameters of the suggestion.

1. Constants defining the possible types of suggestions. As a part of this project, I plan to implement 2 types of suggestions. But broadly, they will be of three types -- content changes, translation related and create new questions/collections/explorations or other newly added entities. In the future if a new distinct category of suggestion is going to be added it could be done so.
2. Constants defining the possible statuses of a suggestion: For the moment, we add 5 possible statuses
  - **In review** - This denotes that the suggestion is in the review process or that it is marked for review. This will be the default status for newly created suggestions.
  - **Rejected** - The suggestion has been reviewed and the changes have been rejected.
  - **Approved** - The suggestion has been reviewed and the changes have been approved. This is analogous to giving a LGTM approval.
  - **Accepted** - The suggestion has been reviewed and changes have been merged.
  - **Invalid** - The suggestion has been invalidated due to some conflicting changes made since the suggestion was created. Such a suggestion cannot be merged. Possible scenarios include, changing state content for a state that doesn't exist anymore, the content being translated has been changed since when the translation was suggested and hence the translation is wrong and would need to be redone.
3. For each type of suggestion defined above, we need a list of possible suggestions that fall under that type. Adding a new suggestion to the framework would require you to define an appropriate constant under the appropriate type of suggestion. This will be a dict where the keys are different types of suggestions and values are a list of sub-types for suggestions of that type.
4. (optional) For each of the above defined possible suggestions sub-types, we define what would be the minimum role a reviewer would need to be able to review the suggestion. (In general this should be the newly added reviewer role, but for some beta features, it would be nice to restrict the permission to just admins).

The suggestion model should have the following properties.

- The type of the suggestion: A value which is one of the constants defining the types of suggestions.
- Suggestion sub-type: This will be a value from the list of possible suggestions (constant defined above) for the given suggestion type. This parameter helps us identify what specific parameters need to be passed in to the created suggestion.
- Status of the suggestion: Will indicate the status of the suggestion. By default the status will be set to “In review” if the suggestion needs to be reviewed before accepting it. This parameter will be one of the defined constants for statuses of a suggestion.
- Category of suggestion: Will denote the category which the user will be scored on. If it is not provided, the user will not be scored.
  - For suggestions of translation type, this parameter will store the language.
  - For suggestions of content type or creating new entities, this parameter will store the subject category (Algebra, Algorithms, etc).
- Author of the suggestion: Stores the user ID of the author.
- Reviewer who approved the suggestion: It would be good to know who approved the changes for future reference.
- A Feedback thread ID: When the suggestion is created, this will be null. If the reviewer wants to discuss aspects with the author of the suggestion, a thread will be created dynamically.
- Assigned reviewer: If any reviewer is specifically assigned to review the suggestion, this parameter would be set to that user’s ID. This parameter is not required.
- Suggestion parameters: This field will be a JSON object whose parameters depend on the suggestion sub-type. This parameter is specific to what the suggestion is made for.


Each suggestion will need to have the following functions.

- Function to check validity of a suggestion.
- Function to convert the suggestion to a dict.
- Function to change the status to rejected or approved.
- Function that accepts the suggestion. This function validates the suggestion, and then if the suggestion is valid, it will accept the suggestion. For suggestions that create new entities, the appropriate creation function should be triggered. For content and translation related suggestions, a call to the appropriate `apply_change_list` function should be made to actually make the changes. For newly added suggestion names, the appropriate function should be called in this function. If the suggestion is not valid, the status will be set as invalid.

## The suggestion parameters field

The suggestion parameters field will be dependent of the sub-type of the suggestion. Here I outline the required parameters for a few of the possible suggestions.

- For content changes or translation changes, 3 parameters would be required
  - Lesson ID: This parameter stores the ID of the exploration/collection/question. When new components like skills, topics, stories, etc are added, their IDs can also be stored in this parameter.
  - Lesson version number: This parameter will store the version number of the exploration or collection where the change is being suggested at the time the suggestion was created. It is required to validate that the suggested change is still valid before accepting (merging) the suggestion.
  - A change list linked with this suggestion: For a start, this could just be one element to keep suggestions linked to exactly one type of change. This will follow the same structure as the change lists currently passed into the `apply_change_list` function present in explorations, collections and questions. The dicts in this list should contain:
    - The new value of the content.
    - Based on the command type, the remaining elements of the change list are decided.
    - The command linked with this suggestion: This parameter specifies the type of command that needs to be performed by the change list. The defined constants in the codebase allow editing of the contents of the lesson by using a `change_list` (commands defined as `CMD_*` in the `*_domain` files for explorations, collections and questions). Some examples of the changes possible using this method are listed below:
      - Change exploration properties like title, objective, tags, etc.
      - Change type of interaction, answer groups, hints, solutions, and other state properties for an exploration.
      - Add or remove nodes from a collection, change required skill IDs, acquired skill IDs, etc.
      - Edit question data, linked skills and title for questions.
      - Translations are always linked to a `subtitledHtml` component which comprises of concept card, hints, solutions, etc. By using the `edit hint` command, the translation linked with it can be updated.

- 
- **Note:** Not all of the above supported changes will need to be exposed as a part of the suggestions framework. Only the relevant ones will be added based on demand from users. In this project we take a look at 2 of the allowed commands, in particular, suggesting edits to content, and suggesting edits to answer groups.
  - For creation of new entities, 2 parameters required are:
    - The type of entity being created
    - A dictionary with all the details that need to be passed to the function that creates the entity.
  - To add/update roles for users:
    - The user ID to update roles for.
    - The new role to be assigned to the user.

## The review system

### User scoring system (A basic Idea)

To maintain quality of reviews, a scoring system will be implemented. When a user makes a contribution (gets a suggestion accepted), he gains one point for that type of suggestion. The values that need to be stored in the userSuggestions model are:

- User ID: The user whose score is recorded.
- Score.
- The category of the suggestion.
- The type of suggestion: This along with the category gives us details of the domain in which the user has contributed with the above score.

The relevant functions to be added include

- Increase score by 1: To be called when a contribution is accepted.
- Check if user's score is above 10 (the set threshold).

### How this works

Each user will start out with 0 score. Any user regardless of their score can make suggestions. As and when their suggestion gets accepted, their score for the category increases by 1. Once their score is above the set threshold, they are given the reviewer role and can help out in reviewing suggestions where their score is above the threshold.

Users with scores above the threshold will be allowed to approve or reject any suggestion in that category (not allowed to accept). These users will be termed as being part of the *global reviewer pool for \**, where \* is the category.

### Trusted reviewers

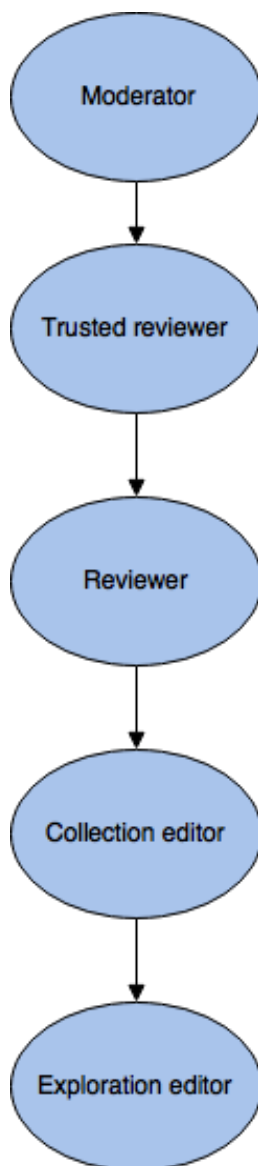
These are users that will be given merge access for an exploration. Each exploration will have its own set of trusted reviewers. The exploration owners will be able to add users to the exploration's trusted reviewer list. These users will be given a role which allows them to accept suggestions to explorations that are not owned by them.



## Review system

Two new actions should be defined for accepting changes to activity and to approve a suggestion thread.

Two different levels of reviewers will be introduced to aid the review process. A trusted reviewer will have permissions of an exploration editor along with permission to accept changes to activities that are not owned by the user. If the user has a score of at least the threshold for any category, the user should be given the reviewer role which allows the user to approve a suggestion (in that category) put forth to any activity. The updated role hierarchy would look like:



A reviewer would be allowed to use the action:

- Approve a suggestion.

A trusted reviewer would in addition be allowed to use:

- Accept a suggestion (in effect, allows a user with this role to accept changes to activities owned by someone else).


### How the review process works

- For content related suggestions to an exploration:
  - The suggester makes a suggestion.
  - A user with the reviewer role (with enough score in the category of the suggestion) can approve this suggestion.
  - A user who is a trusted reviewer can approve as well as accept this suggestion.
- For translation related suggestions to an exploration (an initial idea):
  - The translator adds a translation.
  - Any user with enough score for the language can review the translation
  - The merge rights are still reserved for the trusted reviewers for the exploration.
    - After the basic review system is set up, another role can be made for *trusted translators*. This role however will need to be characterized by the languages the trusted translator will be allowed to accept translations for.
    - Another in demand feature request for audio translation reviews was to add a rotational default reviewer for any incoming audio translation. This can also be set up after the basic system is in place.

### Viability

This is a good model in the long run. However in the short run, This may not work out well as all users will have 0 score. So we need a good strategy to kickstart this scoring model. Here are two possible approaches, both approaches may be carried out simultaneously too.

**Possibility 1:** Initially, the responsibility is on the creators to assign trusted reviewers for their owned explorations. These trusted reviewers will aid in the review process to allow other users to get more and more suggestions accepted. Once we have a sufficient number of suggestions accepted, the review process will work smoothly as there will be sufficient number of reviewers for most categories.



**Possibility 2:** We keep the required threshold score low initially, this will allow us to get in reviewers quickly and once the reviewer pool grows significant, we raise the threshold to a realistic value. This approach would help kickstart the process of reviewing, but may compromise on quality of reviews. The *cost* vs the *benefits* should be compared before this is implemented.

## Playbook to add handle a new suggestion sub-type

The following steps need to be followed to add a suggestion of a particular type.

1. First, identify the function that needs to be called in order to complete the suggestion. In the situations that have been described in this proposal, this step requires you to identify that for suggestions that change exploration content, the `apply_change_list` function needs to be called. For adding a new question, the `add_question` function needs to be called.
2. Then identify what values (the `suggestion_parameters` field) would be required to perform the command. This is obtained by looking at the parameters that the above function takes in.
3. Add a new constant in the dictionary containing the sub-types for the type of suggestion.
4. Add proper validation checks to see if the suggestion is valid (will be run before accepting the suggestion). Here you will need to validate the `suggestion_parameters` field so that the necessary values are all present. Any extra validation is also performed in this step.
5. Add an extra case in the `accept_suggestion` function to handle this new type of suggestion. In the `accept_suggestion` function, the suggestion is validated and then a call should be made to the appropriate function identified in step 1 by passing all the required parameters identified in step 2.

To add a new type of suggestions

1. Add a new constant alongside other suggestion type constants.
2. Define an empty list of suggestions that fall under this category.
3. Now new suggestions can be added to this new type using the steps mentioned in the previous section.

## Examples of suggestion implementations

This section gives examples of suggestions and how they can be implemented in the suggestion framework.


- Edit the content of an exploration state, including the concept card, answer groups, hints, solutions, etc.
  - This suggestion will be linked to CMD\_EDIT\_STATE\_PROPERTY command for an exploration.
  - The type of the suggestion would be set as “content changes”.
  - The category of suggestion would be linked to the subject category of the exploration.
  - The lesson ID will be set to the ID of exploration where the edit is being made.
  - The change list would look like:
 

```
[[
    Cmd: CMD_EDIT_STATE_PROPERTY
    Property_name: STATE_PROPERTY_*,
    State_name: 'First state',
    New_value: some new content
  ]]
```
  - The suggestion properties would contain the above mentioned exploration ID and the change list and the exploration version.
  - This suggestion will need to be reviewed before it can be merged.
  - Any reviewer with a score above the threshold for the subject category can state approval or rejection.
  - The trusted reviewer can then accept the changes if the suggestion has been approved, or if he approves the suggestion himself
  
- Adding a Question
  - This suggestion will be of type “create new entity”.
  - The suggestion parameters would include
    - New question dict containing title, schema version, language code, question data, etc (the exact dictionary that will be passed to a function that creates a question).
    - Entity type as “question”.
  - The appropriate reviewer for this situation would be either an admin or moderator as the question is an independent entity. In the future, there

could be a pool of sitewide questions reviewers, similar to the translation reviewer pool mentioned below.

- Once approved this question can be added.
- Adding translations for an exploration component (say, hints).
  - This suggestion will be linked to CMD\_EDIT\_STATE\_PROPERTY command for an exploration.
  - The type of the suggestion would be set as “translation related”.
  - The category of suggestion would be linked to the language of the translation.
  - The lesson ID will be set to the ID of exploration where the edit is being made.
  - The change list would look like:
 

```
[{
  Cmd: CMD_EDIT_STATE_PROPERTY
  Property_name: STATE_PROPERTY_INTERACTION_HINTS,
  State_name: 'First state',
  New_value: {
    Html: same hint text as before,
    Audio_translations: {
      'en': {
        'Filename': newly_uploaded_filename.mp3,
        'File_size': file size of the above file,
        'Needs_update': true
      }
    }
  }
}]
```
  - The suggestion properties would contain the above mentioned exploration ID and the change list.
  - This suggestion may or may not require reviews.
    - If this suggestion is created by a user who has score above the threshold for the language in which the user added the translation, the user will be given an option on whether or not this translation should be reviewed before adding it. Typically, we would like to have the translation reviewed before it is added (This was Anmol’s opinion. He mentioned that when adding a lot of translations, it is possible that you mess up a few of them in between. It could be a very negative

- 
- impact on the learner if wrong translations are played, so it would be better to have it reviewed). However, for situations where the translations need to be added in a short span of time, the translator should be given a choice to directly add the translation without review.
- If the creator of the suggestion has a score below the threshold, the translation is submitted for review.
  - If the user chooses to not opt for a review, this suggestion would directly be approved. Otherwise, once the suggestion is approved by a user with score above the threshold for the language, it can be merged into the exploration.
- Adding a demonstrative image for the exploration
    - The image will be a part of any of the content elements of the exploration.
    - So this is again a content change suggestion.
    - This reviewers for this suggestion would be the trusted reviewers of the exploration where this image is added
    - The element where the image is added will have a changed HTML and a change list will be present in the suggestion properties. The appropriate function will be called to apply the change list.
  - Add an answer to training data (Can be handled in the future)
    - The appropriate function to be called is identified.
    - A new type of suggestions will be created for adding values to training data.
    - Required values for training will be passed through the suggestion properties.
    - A check pertaining to this type of suggestion will be added to the validate suggestion function.
    - The appropriate function is called in the accept\_suggestion handler when such a suggestion is created by a user.

## Milestones

### Add a general structure for a suggestion (Milestone 1)

As a part of this milestone, the suggestion model will be defined and relevant domain objects, services and controllers will be added. Finally we will replace the existing suggestion framework to use newly added framework. We will hook up the existing frontend views to the newly added controllers instead of the suggestion controllers.

Breakdown of suggestions to be added in this milestone

- Add the new model pertaining to suggestions in `storage/suggestions/gae_models.py`.
- Add a domain object for the newly added model in `domain/suggestion_domain.py`.
  - A set of constants should be defined as mentioned in the overview section.
  - Functions will need to be defined to create a suggestion, validate a suggestion, update a suggestion (if the suggestion hasn't been accepted yet), get author name, validate a suggestion.
  - A function to get the feedback thread object linked to the suggestion should also be written.
  - Backend tests will need to be written for all the above functions
- Suggestion services will be added under `domain/suggestion_services.py`
  - Functions to create a new suggestion and update an existing suggestion (which is "in review") need to be added.
  - Functions need to be written to query suggestions by category, type of suggestion, status of the suggestion and author of the suggestion.
  - Functions to accept, reject and approve a suggestion should be written.
  - Each of these functions will need backend tests
- Suggestion controllers that will need to be added under `controllers/suggestion.py`
  - SuggestionHandler that handles creation and updating a suggestion
  - SuggestionListHandler that will handle get queries for suggestions. Various cases here would include
    - Get all suggestions linked to an author
    - Get all suggestions in a particular category based on status (In review, approved, rejected or accepted)
    - Get all suggestions linked to a lesson (So that it is consistent with the current scenario where all suggestions for an exploration can be retrieved).
  - SuggestionActionHandler that will be responsible for changing status of the suggestion.
- Make new routes to access the newly added controllers.

- A new integration test will be added to test the newly added suggestion framework. This test should
  - Create an exploration, create a content related suggestion to change the state content (the concept card value).
  - Accept the suggestion and the changes should be visible in the exploration
  - Try another suggestion to modify the content of the state and reject it this time
  - No changes should be made to the exploration.
  - After the reviewer access control is defined in the next milestone, this test should be modified as only trusted reviewers can accept the changes.
  - Also a scenario where the initial suggestion is updated before being accepted should be tested.
- As a part of this milestone, minimal frontend changes will be made. Suggestions are created in the current suggestions framework by just creating a post request to the newly created SuggestionHandler with the values required.
  - This will be changed to create a suggestion by passing the additional parameters and command type equal to `CMD_EDIT_STATE_PROPERTY` and property type equal to `STATE_PROPERTY_CONTENT`. (File to edit: `pages/exploration_player/LearnerLocalNav.js`)
  - In the creator view where the threads are shown and can be accepted or rejected, changes will be made to use the new suggestions framework. (Affected files: `pages/exploration_editor/ThreadDataService.js`, `pages/exploration_editor/Feedback.js`).
  - After linking it up to the new framework, the protractor test should continue to work. This will be used as one of the checks to see that the suggestion system actually works as expected.
    - Test will be modified to test a situation where the suggestion is updated before it is accepted
- Write a one-off job to convert all “old” suggestions to “new” suggestions with command type equal to `CMD_EDIT_STATE_PROPERTY` and property type equal to `STATE_PROPERTY_CONTENT`. This will be added to `domain/feedback_jobs_one_off.py`. Write a test for the same. This step completes the migration to the newly added suggestion framework. The structure of the newly added suggestion would look as shown below.

```
{
  suggestion_type: SUGGESTION_TYPE_CHANGE_CONTENT;
  suggestion_name: CHANGE_STATE_CONTENT;
  status: same as the status of the suggestion;
  category: the subject category of the exploration;
  feedback_thread_id: Get the ID of the linked thread;
  author_id: suggestion.author_id;
```



```
suggestion_parameters: {
  lesson_id: suggestion.exploration_id;
  lesson_version: suggestion.exploration_version;
  change_list: _create_change_list_from_suggestion(suggestion);
}
}
The change_list would look like:
[[{
  'cmd': exp_domain.CMD_EDIT_STATE_PROPERTY,
  'state_name': suggestion.state_name,
  'property_name': exp_domain.STATE_PROPERTY_CONTENT,
  'new_value': {
    'html': suggestion.suggestion_html,
    'audio_translations': {
      'en': translation dict
    }
  }
}]
```

## Reviewers (Milestone 2)

### The scoring system (Milestone 2.1)

- First we define the userSuggestionsModel in storage/user/gae\_models.py. The parameters will be as stated above for a basic scoring system.
- Define the appropriate class in domain/user\_services.py to create the domain object.
- Add functions to do the following:
  - To get score of a user for a given category and type of suggestion.
  - To update the score of a user for a given category and type of suggestion (For the basic model, update score by one).
  - Check if user has score above the threshold for a given category and type of suggestion.
  - Given a user ID, get all records where the score is above the threshold for the user.
  - Write tests for all the above implemented functionality.
- Edit the previously added accept suggestion function to add a point to the author of the suggestion for the given category and type of suggestion.

Now we implement the new role hierarchy for reviewers and trusted reviewers.

## The review system (Milestone 2.2)

The two new roles will be added to the present role hierarchy for reviewers and trusted reviewers.

- First we add two new role constants in feconf.py
- Now we add the roles as per the stated hierarchy in domain/role\_services.py
- Then the two new actions in domain/role\_services.py are added. Add them to the allowed actions for the newly added roles.
- Another category of users called trusted reviewers will be added in domain/rights\_manager.py. These will be users who are trusted reviewers for the activity. These users will have the newly added trusted reviewer role. A model change would be necessary in storage/exploration/gae\_models.py (related model: ExplorationRightsModel).
- A function should be added to check if user can accept suggestions linked with this activity. The user should have the *accept changes from suggestion thread* action (newly added) in their list of actions. Additionally the user should be either owner, editor or trusted reviewer for the activity.
- A function should be added to check if user can approve/reject suggestions linked with this activity. For this having the reviewer role is sufficient.
- A function should be added to check if user can edit a suggestion that is in review. This permission will be available to trusted reviewers, editors, owner or the user who created the thread.
- These newly added permissions should be tested.
- New decorators should be added to domain/acl\_decorators.py for the following:
  - Can user accept changes to activity.
  - Can user accept suggestion.
    - If the suggestion is not linked to an activity, in general restrict this permission to admin.
  - Can user approve suggestion.
    - If the suggestion is linked to an activity, check using the above implemented function.
    - Else check the minimum role needed to review the suggestion from suggestion\_domain.py.
  - Can user edit a suggestion that is in review.
- Relevant tests for the decorators should be added.
- Now use the appropriate decorators in the suggestion handlers present in controllers/suggestion.py.

- Edit the integration test written to test the newly added permissions also. The complete flow should be tested
  - The suggestion is created by a user.
  - Then the suggestion is approved by a reviewer.
  - Then the suggestion is accepted by a trusted reviewer.
  - Intermediate steps can be added to test editing the suggestion before merging, accept another suggestion that makes the current suggestion invalid and then the suggestion should be marked invalid, etc.
  - Also once the suggestion is accepted, the user score should have increased.
  - Some other situations worth testing include:
    - Try accepting the suggestion with users who don't have enough permissions.
    - Try approving the suggestion with users with reviewer role but don't have enough score for the category.
  - Any non trivial situation that pops up while testing the framework will be added as a test in the integration test if applicable.
- A change will be made in the `create_suggestion` function created in milestone 1. If the user has enough permissions to accept the suggestion that the user is creating, an extra parameter can be passed in to bypass the review process. In the `create_suggestion` function itself, the appropriate permission check would be done and the suggestion would be approved as soon as it is created.

Now we change parts of the frontend to allow the review system to work.

- For this first we provide the option for exploration admins to give out trusted reviewer roles to users. For this we add an extra option for trusted reviewers in the edit roles dropdown.
- The create suggestion dialog will contain a checkbox that can be ticked to bypass review if the user will be allowed to. A warning message will be displayed to the user stating that it is not recommended to bypass the review system and should be done if it is truly urgent to get this suggestion in.
- The current view suggestion modal will be edited to include a button for "approve" when the user viewing the suggestion is a reviewer.
- The protractor test will need to be modified to assign proper roles to the users before trying to accept or reject changes. Also the test should be modified to also include an intermediate step to test approving a thread before accepting. Some corner cases that can be tested here include
  - Trying to approve a suggestion with users with and without permissions to do so.
  - Trying to accept suggestions with the same users.
  - Test the new functionality to bypass the review system with different users.

**Note:** Adding permissions for translation based reviewers can be implemented as an extension of this review system. As per the proposed model, users with the reviewer role and a good enough score in the language will be allowed to approve the translation suggestion for an exploration. The merge access is preserved for trusted reviewers (Allowing only trusted reviewers to merge a translation suggestion may not be very meaningful from a UX perspective, But to avoid overlaps with the translation dashboard project, I didn't want to go into details for the review process of a translation suggestion. Once a translator role is introduced for an exploration, a user with the translator role would be the ideal person to handle translation related suggestions for that exploration, though the global pool of reviewers for a particular language can state approval regarding the same.)

## Add the backend for 2 new suggestions (Milestone 3)

As a part of this milestone we will implement two new types of suggestions. This will help us ensure that the system that we built indeed generalizes to various kinds of suggestions. The two suggestions for this milestone have been chosen based on the priorities mentioned by Sean. First of them would be a suggestion to add a question, and the second to suggest changes to answer groups.

Both of these suggestions can be implemented using the same set of steps (the same steps mentioned in the playbook). The major parameters that will be considered are mentioned below.

### Add a new question

- Function to be called when suggestion is accepted: `add_question` in `question_services.py`
- Parameters to be passed in `suggestion_parameters`: a question dict that has the data of the question that will be added.
- A new type of suggestion needs to be created. This type would involve all suggestions relating to creation of new entities.
- Validation: The question dict should have all the necessary parameters for creating a new question. This can be validated by creating a domain object and calling `validate()` on it. But as the `add_question` also checks the validity of the object, just checking that the required parameters are passed should be enough.

### Suggest changes to answer groups

- Function to be called when suggestion is accepted: `update_exploration` in `exp_services.py`

- Parameters to be passed in `suggestion_parameters`: the exploration ID, exploration version and the `change_list`. The structure of the change list is determined by investigating the `apply_change_list` function.
- This suggestion will be a part of the changes to content type which was added as a part of the first milestone.
- Validation: Firstly, the state where the answer groups are edited should exist in the exploration. Then the data passed in must be checked to make sure all the required details are present. Rigorous validation of the details of the answer groups list is performed in the `update_interaction_answer_groups` function, so they will not be replicated here.

#### Optional suggestions that can be added (If time permits)

- Suggest changes to hints.
- Suggest changes to solutions (These two suggestions will follow a similar implementation as the changes to answer groups suggestion).
- Add a question to a skill (A possible overlap with the skills project, can collaborate and help build a review based system to add questions to skills.)
- Add new audio translations (Again, a possible overlap with the translations dashboard project. Can collaborate to make the translations dashboard feature also link to the review system).

## Future extension projects (will be taken up after completion of the 3 milestones)

### Build a suitable UI to allow suggesting edits to answer groups and to add questions (Important!)

Once the respective backends are implemented for these suggestions as a part of the final milestone, the next course of action to allow users to use this functionality is to make suitable UIs for being able to create and review these suggestions. This will be taken as an extension of the third milestone if time permits. This will allow this functionality to be used by the community so that we can obtain feedback and hence improve the framework.

### Delink Feedback threads from explorations (Important!)

As mentioned in the implementation details, for the review system to become truly general, feedback threads need to be delinked from explorations. Two possible approaches were outlined above. This step has to be completed to allow generalizing the suggestions framework and the review system to be comment on any suggestion thread to give feedback on any suggested change.

### Develop an extension of the review system for translations (Important!)

Reviewers for translations is a complex domain. There should be a set of users who can be called the *translation team* who have rights to add, review, and change translations for any exploration. This is partly possible with the basic implementation of the review system. The add and change steps had to go through the review step and the reviewers would need to have a high enough score for the language in order to approve. This is fine, except, the trusted reviewer for an exploration was finally needed to complete the process and accept the translation.

This could be a meaningless step for translation suggestions. The trusted reviewer may not be proficient in the language that the translation is in and hence there is no reason for him to be the one to accept the translation. Instead these rights need to be provided to either the translators for the exploration, or to a global set of “trusted translators”, or both of them. This project extension is possible but relies on the translation dashboard to be implemented to have translators for an exploration.

## A reviewer dashboard (Better UX)

For better UX for reviewers, a dashboard can be created to show users all of the suggestions that they are assigned for review. Also in general they should view all suggestions in categories they are allowed to review for. The user can be presented with an interface to filter the suggestions based on various relevant parameters so that it will be easy for the reviewer to find a particular set of suggestion to review.

## Make a rotational default reviewer assignment for all newly created suggestions (Feature request from the audio team)

As a part of the basic review system that was implemented, a default reviewer can be assigned by the creator. But as the number of suggestions increase, this could become a very cumbersome process. So the trusted reviewers can be assigned rotationally as default reviewers for the incoming suggestions.

## Add suggestions pertaining to newly added functionality

As more and more functionality is added into Oppia, different types of lessons are created, etc, the community should also be allowed to contribute in these new areas. For this new suggestions need to be added into the framework. For instance, suggestions pertaining to audio translations could be added after the translation dashboard is set up, suggestions pertaining to adding new skills, adding questions into a skill, etc can be added once these features are added. For each of these suggestions, their relevant UI component must also be added so that the user can create such suggestions.

## Generalise the process of suggesting edits to an exploration

Once a learner is allowed to create suggestions to suggest edits for various parts of an exploration, a good UI design must be implemented so that the learner can easily (and without getting confused) select the component that needs to be edited and give its new value.

## Timeline

Dates	Tasks that will be completed
Community bonding period: April 23rd - May 7th (2 week)	Finish ongoing PRs, communicate with my mentor to make any required changes to the implementation plan, and complete responsibilities for May release. Also my college final exams would be going on.
Coding period starts (on 14th May) May 10th - 15th	Implement the required model and domain object for the suggestion model, add the related services, and add relevant backend tests for the same.
May 16th - May 23rd	Add controllers for the suggestions model, new routes will be created for the same, relevant tests will be added.
May 24th - May 28st	A new integration test will be added to test the above added framework
May 29th - June 2nd (Release cut on June 2nd)	The related frontend changes will be implemented in order to use the new framework
June 3rd - June 7th First evaluation	A one off job will be written to migrate the suggestions from the old framework to the new framework.
June 7th to June 11th	Buffer period for the first milestone. Manual testing will be done to catch any bugs. If any bugs are reported, I will work to fix them.
June 11th to June 15th	The existing suggestions model will be removed from the codebase, all related controllers, services and domain objects will be removed.



June 16th - June 20th	The new user scores model will be defined, all related domain objects and services will also be added. Relevant tests will be added.
June 20th - June 27th	New roles are added for trusted reviewers and reviewers, their allowed actions are defined. Related functions to check various permissions are added. New decorators are added. Rigorous tests will be written to test the decorators. Use the decorators in the respective suggestion controllers implemented in milestone 1.
June 28th - July 3rd	Edit the integration test built to also use the various permissions defined for the users. Various corner cases will be tested to make sure that the permissions are correct.
July 3rd - July 5th	Minor changes to frontend to allow assigning the new roles and also to allow reviewers to view the feedback thread. The "approve" button will be added to the view suggestion modal.
July 5th - July 9th (release cut on July 7th) Second Evaluation	Buffer time for milestone 2. Manual testing will be done for the review system that has implemented. Any bugs that surface will be worked upon.
July 9th - July 12th	Clean up any bugs that surfaced during release testing. Any such situation will be added as a test in the unit/integration test as applicable.
July 13th - July 20th	Add a new suggestion type to add a new question, add related validators, also test this new suggestion in the integration test
July 21st - July 26th	Implement a second type of suggestion to change the answer groups. This being similar to the suggestion type implemented

	in milestone 1, an integration test would not be required. However, unit tests will be added for the added code
July 27th - July 31st	Thorough testing of the components added in all 3 milestones. Any bugs reported will be worked upon. Relevant documentation will be added describing the suggestions framework, review system and also the playbook to add a new suggestion will be added.
Aug 1st - Aug 6th (release cut on Aug 4th)	Buffer period for the final milestone. If all tasks are completed so far, an (optional) additional suggestion will be added to allow suggesting changes to hints/solutions.

## Summer Plans

### Which timezone(s) will you primarily be in during the summer?

I will be in India throughout the summer (Timezone: UTC+05:30)

### How much time will you be able to commit to this project?

Between May second week to July end, I will be able to spend about 8-10 hours during weekdays and about 3-4 hours on weekend. Overall per week, I will be able to spend about 55-60 hours on this project.

In August, my classes begin and hence the time spent during weekdays would be lesser (around 3-4 hours) while the weekends I can spend more time (about 8-10 hours, to make up for any pending tasks for the week).

### What jobs, summer classes, and other obligations might you need to work around?

I have no other commitments during the summer. I have no other obligations till July end. In August I will have to get back to college, so the number of hours per week may be slightly lower. I will try to make up during weekends.

## Communication

What is your contact information, and preferred method of communication?

My contact information:

Mobile number: +91-7760351907

Email ID: [nithusha21@yahoo.co.in](mailto:nithusha21@yahoo.co.in) (primary), [nithesh2108@gmail.com](mailto:nithesh2108@gmail.com)

Github tag: @nithusha21

I am also active on Gitter and Hangouts.

How often, and through which channel(s), do you plan on communicating with your mentor?

I plan to maintain a daily record of my progress. Meetings with the mentor can be conducted once or twice a week. I will be in constant touch with my mentor on Hangouts/Gitter. To have meetings, any online service is fine by me. But seeing that all my meetings with the Oppia community have been on Hangouts so far, we'll probably use Hangouts for meetings.

## About me

I'm Nithesh N. Hariharan, a second year undergraduate studying electrical engineering at Indian Institute of Technology, Madras (India).