

# SWE Project 1 Bonus

## Team 14

Abhijith Anil - 2020101030

Jatin Agarwala - 2020111011

Pratyay Suvarnapathaki - 2020111016

Sneha Raghava Raju - 2020101125

Yash Mehan - 2020111020

## Task 1E: Transition System

### User Management

The user management system has two types of users: user and admin. The admin can create new users (i.e register users), update details, and delete them. Users can log in/log out of the system and update their details.

(Other user functionality like adding music and connecting to Last.Fm is mentioned in other subsystems)

#### X: Set of States

$X = \{ \text{Unregistered, Logged In, Logged Out, Admin} \}$

Unregistered: The user has not been created/registered.

Logged In: The user has logged into the system.

Logged Out: The user exists (has been created) but is not logged in.

Admin Logged In: The admin is logged into the system.

#### X0: Initial State

Logged Out - The user is not logged into the system.

#### U: Action Space

Login - The user (user/admin) enters their username and password to login to the system.

Logout - The user logs out of the system.

Register User - The admin creates a user by entering the necessary details (username, etc).

Delete User - The admin deletes a user from the system.

Update User - The admin updates the details of a user (eg: username).

#### Y: Output Space

User Created - A new user is created with the entered details (username, password), and added to the user list.

User Logged In - The user is successfully authenticated and able to access the system; session token created.

User Logged Out - The user is logged out from the system; session ends.

User Deleted - User no longer exists and is unable to access the system.

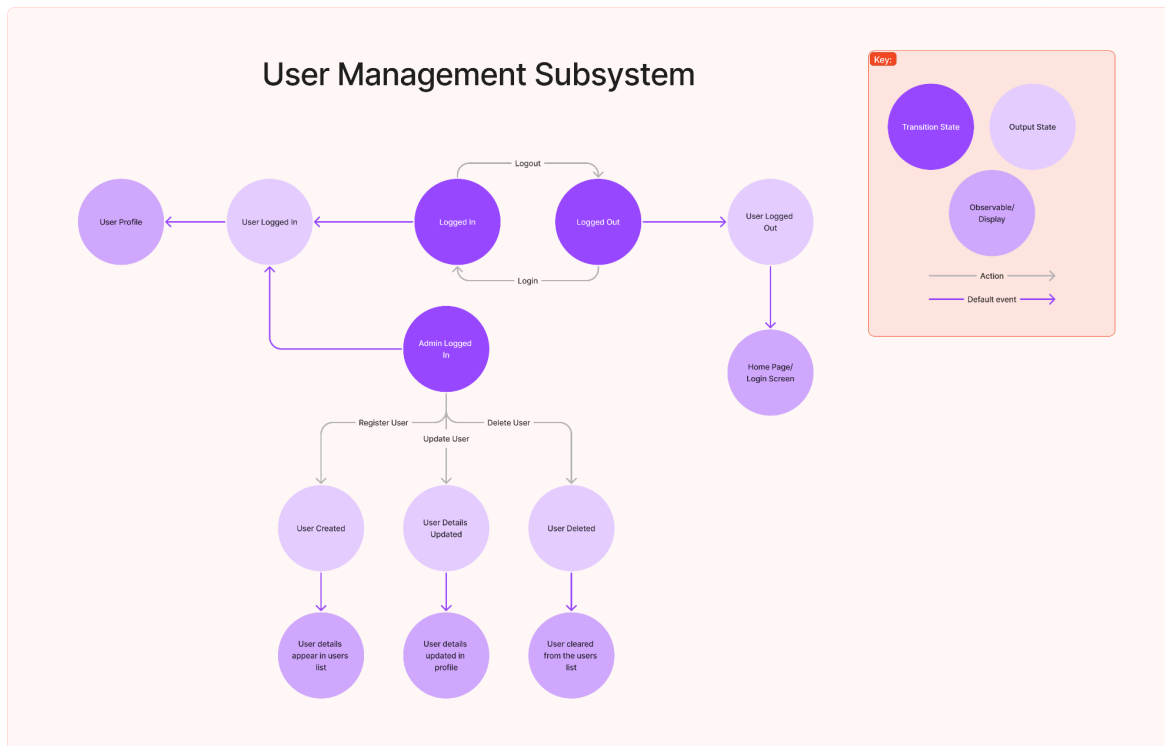
User Details Updated - The new details (eg: username) are updated for that user, and can be used to login.

#### **f : Transition Relation**

<b>State</b>	<b>Action</b>	<b>Next State</b>
Logged Out	Login	Logged In
Logged In	Logout	Logged Out
Admin Logged In	Register User	Admin Logged In
Admin Logged In	Update User	Admin Logged In
Admin Logged In	Delete User	Admin Logged In
Logged In (User)	Update User	Logged In (User)
Unregistered	Register User	Admin Logged In

#### **h: Display Map**

<b>State</b>	<b>Observable</b>
Unregistered	User Logged Out - the user does not have access to the system, only sees the home page
Logged In	User Logged In - User can view their profile, song library, etc.
Logged Out	User Logged Out
Admin Logged In	User Logged In - the admin can access and edit the song library, create and manage users, etc.



## Library management - TS Model 1

### Initial State

`import` button idle

### State space

- `import` button idle
- Waiting for import type from user
- `downloadAudio()` initiated for AppContext instance
- `importAudioQueue` is empty
- `importAudioQueue` is not empty
- import initiated
- import completes gracefully
- import terminates with error
- importing killed

### Action Space

- User clicks `import` button
- User selects import type
- request sent to backend
- push import request to `importAudioQueue`
- `isRunning()` returns true
- `isRunning()` returns false
- request popped from `importAudioQueue`

- initiate `retryImportAudio()`
- user sends kill import request

### Output Space

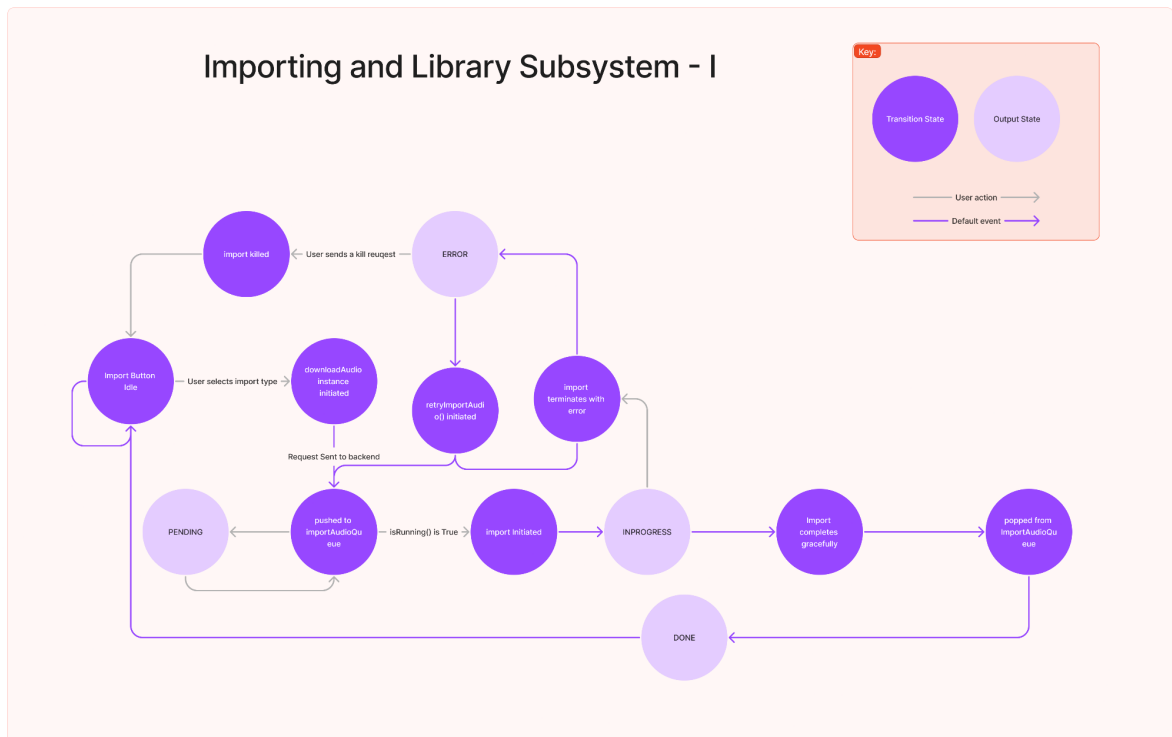
- PENDING import Status
- INPROGRESS import Status
- DONE import Status
- ERROR import Status

### Transition Relations

Initial State	Action	Next State
`import` button idle	-	`import` button idle
import button idle	User selects import type	`downloadAudio()` initiated for AppContext instance
`downloadAudio()` initiated for AppContext instance	request sent to backend	`importAudioQueue` is empty
`importAudioQueue` is empty	push import request to `importAudioQueue`	`importAudioQueue` is not empty
`importAudioQueue` is not empty	isRunning() returns true	import initiated
import initiated	-	import completes gracefully
import initiated	-	import terminates with error
import completes gracefully	request popped from `importAudioQueue`	`importAudioQueue` is empty
import terminates with error	initiate `retryImportAudio()`	`importAudioQueue` is not empty
import terminates with error	user sends kill import request	importing killed

### Display Map

State	Observable
Push import request to importAudioQueue	PENDING
import terminates with error	ERROR
import Initiated	INPROGRESS
popped from ImportAudioQueue	DONE.



## Library management - TS Model 2

### Initial State

- an instance of `CollectionWatchService` is running
- `WatchEventKind` of `CollectionWatchService` is `ENTRY\_CREATE`
- `indexFolder` is called
- `indexNewFile` in called
- `readTrackMetadata` is called
- `scanDirectory` is called

### Action Space

- CollectionWatchService detects a new file in the collection directory

### Output Space

- new track is added to the collection

### Transition System

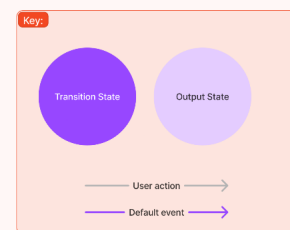
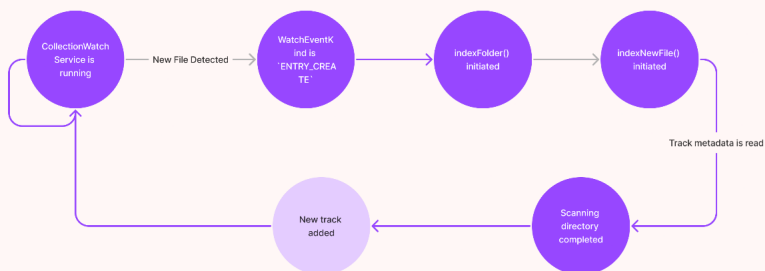
Initial State	Action	Next State
an instance of `CollectionWatchService` is running	CollectionWatchService detects a new file in the collection directory	`WatchEventKind` of `CollectionWatchService` is `ENTRY_CREATE`
`WatchEventKind` of `CollectionWatchService` is	-	indexFolder initiated

'ENTRY_CREATE'		
indexFolder is initiated	-	indexNewFile initiated
indexNewFile initiated	Track metadata is read	Scanning Directory is complete
Scanning directory is complete	-	new track is added to collection

## Display Map

State	Observable
Scanning directory Completes	New Track Added to System

## Importing and Library Subsystem - II



## Last.Fm integration

### X : Set of States

- Last.Fm not linked
- Last.Fm linked
- Not listening
- Listening
- Track Ended
- Unliked
- Liked

### X<sup>0</sup> : Initial state

Last.Fm not linked

### U : Set of Actions



- Link Last.Fm account
- Unlink Last.Fm account

- PlayStarted
- PlayCompleted
- PlayStopped
- PlayPaused
- TrackLiked
- TrackUnliked





#### f : Transition Relations

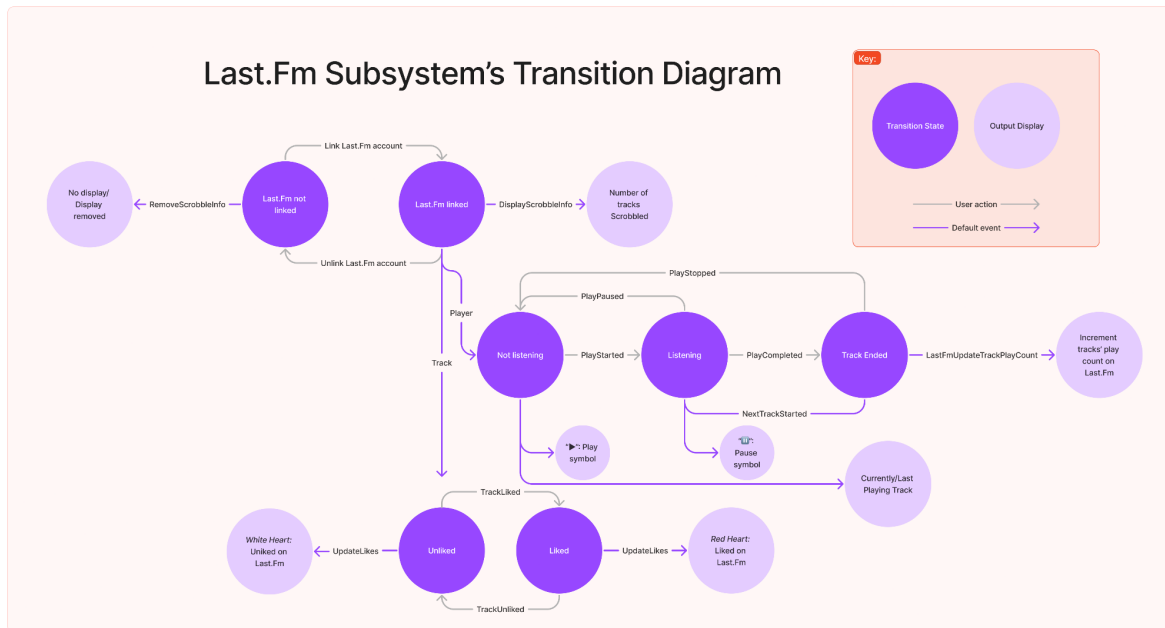
Current State	Action	Next State
Last.Fm not linked	Link Last.Fm account	Last.Fm linked
Last.Fm linked	Unlink Last.Fm account	Last.Fm not linked
Not listening	PlayStarted	Listening
Listening	PlayCompleted	Track Ended
Track Ended	PlayStopped	Not listening
Listening	PlayPaused	Not listening
Unliked	TrackLiked	Liked
Liked	TrackUnliked	Unliked

#### Y : Output space

- Number of tracks scrobbled
- Track Play Count on Last.Fm
- White Heart: Unliked
- Red Heart: Liked
- : Play symbol
- : Pause symbol
- Currently/Last Playing Track

#### h : Display map

- h(Liked): Redheart 
- h(Unliked): Whiteheart 
- h(PlayStarted): Pause symbol 
- h(PlayPaused): Play symbol 
- h(Link Last.Fm Account): [Your account is linked to Last.fm](#)
- h(Unlink Last.Fm account): [Last.fm username and password won't be stored.](#)



## Administrative features

### X : Set of States

- Admin Logged-in
- User Created
- User List
- User Viewing
- User Updated
- User Deleted
- Transcoder Created
- Transcoder Viewing
- Transcoder Updated
- Transcoder Deleted
- Directory Created
- Directory List
- Directory Updated
- Directory Deleted
- Collection Reindexed

### X<sup>0</sup>: Initial state

The initial state is "Admin Logged-in"

### U: Action Space

- Create user
- View user list



- View particular user
- Update user details
- Delete user
- Create transcoder
- View Transcoder
- Update Transcoder
- Delete Transcoder
- Create a music directory
- View a list of music directories
- Update directory details
- Delete directory
- Reindex music files

## F: Transition Relation

Current State	Action	Next State
Admin Logged-in	View user list	User List
User List	Create User	User Created
User List	View particular user	User Viewing
User Viewing	Update user details	User Updated
User Viewing	Delete user	User Deleted
Admin Logged-in	View Transcoder	Transcoder Viewing
Transcoder Viewing	Create transcoder	Transcoder Created
Transcoder Viewing	Update Transcoder	Transcoder Updated
Transcoder Viewing	Delete Transcoder	Transcoder Deleted
Admin Logged-in	View a list of music directories	Directory List
Directory List	Create a music directory	Directory Created
Directory List	Update directory details	Directory Updated
Directory List	Delete directory	Directory Deleted
Admin Logged-in	Reindex music files	Collection Reindexed

## Y: Output space

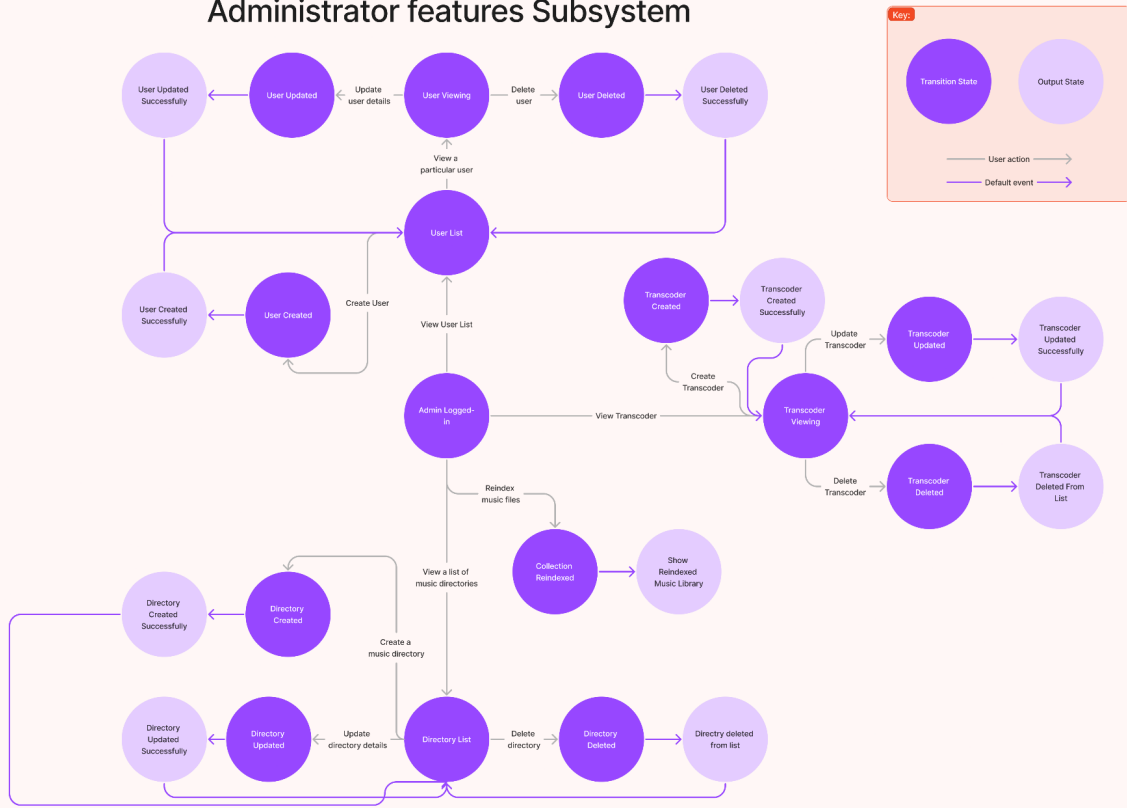
- User Created Successfully : A new user is created with the entered details (username, password), and added to the user list.
- User Updated Successfully : The details of the selected user is updated and is now displayed in the user list.

- User deleted Successfully : The user is deleted from the application and the list.
- Transcoder Created Successfully : A new transcoder has been created, and the information about it is stored.
- Transcoder Updated Successfully : Information about the transcoder is updated successfully.
- Transcoder Deleted From List : A chosen transcoder is deleted from the application.
- Directory Created Successfully : The directory with the path mentioned is added to the list of directories to which the music library is stored.
- Directory Updated Successfully : Details of the directory chosen is updated, and reflected in the list.
- Directory Deleted From List : Chosen directory is deleted from the list of directories.
- Show Reindexed Music Library : After the admin reindexes the library, the updated order is shown.

#### H : Display map

State	Observable
User Created	User Created Successfully
User Updated	User Updated Successfully
User Deleted	User Deleted Successfully
Transcoder Created	Transcoder Created Successfully
Transcoder Updated	Transcoder Updated Successfully
Transcoder Deleted	Transcoder Deleted From List
Directory Created	Directory Created Successfully
Directory Updated	Directory Updated Successfully
Directory Deleted	Directory Deleted From List
Collection Reindexed	Show Reindexed Music Library

## Administrator features Subsystem



# Task 3E: Automated Refactoring

## History and Context

Harman et al. (2001)<sup>[1]</sup> stated that several yet-untractable problems are associated with the elusive art of refactoring. These include, but are not limited to,

- There is usually a need to balance competing constraints.
- Occasionally, there is a need to cope with inconsistency.
- There are often many potential solutions.
- There is typically no perfect answer... but good ones can be recognised.
- There are sometimes no precise rules for computing the best solution.

The same paper also coined the term 'Search-Based Software-Engineering' (SBSE), seemingly opening up the automated refactoring field using optimisation techniques.

Almost concurrently, the domain of software refactoring had also been developing rapidly at the time. Here's a brief history:

- 1984: The notion of 'factoring' is described in Brodie's 'Thinking Forth'.<sup>[2]</sup>
- 1990: Bill Opdyke coins the term 'refactoring' in an ACM SIGPLAN paper.
- 1992: a comprehensive description of 'refactoring' is presented in Opdyke's thesis, 'Refactoring object-oriented frameworks'.<sup>[3]</sup>
- 1999: Refactoring is popularised by Martin Fowler's book of the same name.<sup>[4]</sup>
- 2001: Wide availability of automated aids to refactoring in IDEs for the language Java.

## The State of the Art

The advent of machine learning has made software refactoring faster, more accurate, and more efficient. By automating many everyday refactoring tasks, developers can focus on higher-level design and implementation tasks, improving the quality and reliability of software products.

From this lens, a broad look at the currently established and emerging refactoring tech shows that modern refactoring has been evolving in two 'tracks'.

On the one hand, there exist technologies that can be more or less described as refined variants of the original 'refactoring browser'. Today, these exist as plugins/integrations/addons to popular IDEs.

- The 'Automated Refactoring' section on Wikipedia's page<sup>[5]</sup> on 'Refactoring' exclusively lists tools that fall into this category.

- JetBrains' research<sup>[6]</sup> seems to be focussed on investigating ways to improve their refactoring plugins *within* the exact usability contexts that have existed for decades without straying away from the 'refactoring browser' paradigm.

On the other hand, academia / ML-oriented industry research seems to be exploring the bleeding edge of what can be possibly accomplished by harnessing the power of ML for software engineering. The following are the two prominent technologies currently being explored herein, based on a limited survey of recent literature in the field:

- **Modern Search-Based Refactoring<sup>[7]</sup>**

This technique broadly follows the principles described in the 2001 SBSE paper mentioned above. Firstly, an objective function that measures the quality of the code is defined. This function can be based on maintainability, readability, and performance metrics. Search algorithms then explore the space of possible refactoring solutions. These solutions include changing the code structure, renaming variables, and simplifying control flow. The 'fitness' of each suggestion is evaluated, and the strongest refactorings are finally proposed.

The most popular optimisation strategies used here are Hill Climbing, Genetic/Evolutionary Algorithms, and Swarm Optimization. The paper under consideration, however, does mention concerns regarding the somewhat subjective nature of software quality and whether traditional metrics hold up when evaluating solutions proposed by search-based methods. Furthermore, there are concerns regarding how well the performance of these methods will translate to real-world industry applications.

- **Deep Learning for Predictive Refactoring<sup>[8][9]</sup>**

The first paper under consideration broadly explores supervised machine learning techniques. As it turns out, Random Forests still outperform Neural Networks in most cases, for all 11k+ projects analysed, with the latter only being better for some method-level refactorings. The paper stresses on the importance of metrics such as quantity of commits, lines added in a commit, and number of previous refactorings.

On the other hand, the second paper explores the effectiveness of the Gated recurrent unit (GRU) algorithm. An extension to the paper adds NLP techniques to improve on method and class name suggestions, code comments, and generation of commit messages. This serves as a notable attempt at establishing a truly end-to-end solution.

## Our Proposed Approach

In a broad survey conducted by JetBrains Research<sup>[10]</sup>, quantitative and qualitative analysis showed that:

- Almost two-thirds of developers spend more than one hour in a single session refactoring their code.
- Refactoring types vary significantly in popularity, and...
- Many developers would like to know more about IDE refactoring features but lack the means to do so.

Taking into consideration our findings from the literature survey detailed above, as well as the overall context of how auto-refactoring has organically evolved over the years, we propose the following guidelines for a hypothetical end-to-end auto-refactoring solution:

### 1. If it ain't broke, don't fix it

The 'ends' of the end-to-end are not what people seems to have complaints about. So let's keep it simple. Text is written in the editor of the IDE as usual, and the suggestions made are available in the refactoring menu, which can be accessed by the user at any time. In terms of the user experience, integrating the auto-refactoring functionality should not disrupt the natural flow of one's coding process => all the actual processing will be done in the background.

### 2. Smell Detection

Our hypothetical solution would use a sonarqube-like tool running in the background, to make incremental checks with regards to code quality on the fly. In our experience working on the project, the issue with prospective auto-refactoring isn't the detection of smells, it is the fact that *only detection* is being done. Therefore, the output of this stage will be pushed forward to the next stage in our pipeline. Of course, for maximum interpretability+UX, these may also be made available to the programmer via a 'warnings' panel accessible through the bottom bar in the IDE.

### 3. Churning Out Refactorings

We assume this API-ified version of sonarqube makes 'levels' of smells clear (i.e. whether a smell is at the class level, the method level, or the particular-line-of-code level). Now, based on our literature survey, we would use an ensemble of models, also chugging along as background processes, to offer appropriate refactorings for the level of issue they're best suited to address. A possible system may consist of:

- The existing IntelliJ system / A language model (as explored by Alenezi et al.) to address line-of-code level issues such as identifier names, comments, token placement, etc.
- A search-based optimiser (as described by Mohan et al.) for method-level refactorings.

- A random forest / neural network / GRU / other deep learning based techniques to address larger, class-level refactorings.

#### 4. The Other End

As mentioned above, the final decision of whether a refactoring is to be applied lies with the user. The auto-refactoring system may assist the user by reporting relevant metrics, and propose several refactorings for a particular instance (just as GitHub Copilot currently does) in case several distinct suggestions are similarly 'good'. Of course, there is quite a lot of subjectivity here owing to the different metrics used by the different techniques and the fundamental question of if an improvement in these metrics is truly reflective of good code (a concern echoed across papers in the literature survey).

## References

- [1]: [Harman and Jones \(2001\): 'Search Based software Engineering'](#)
- [2]: [Leo Brodie \(1994\): 'Thinking Forth'](#)
- [3]: [William Opdyke \(1992\): 'Refactoring object-oriented frameworks'](#)
- [4]: [Fowler et al. \(1999\): 'Refactoring'](#)
- [5]: [Wikipedia page on 'Code Refactoring'](#)
- [6]: [JetBrains Research: ML Methods in SWE Lab](#)
- [7]: [Mohan et al. \(2018\): 'A survey of search-based refactoring for software maintenance'](#)
- [8]: [Aniche et al. \(2020\): 'The Effectiveness of Supervised ML Algorithms in Predicting Software Refactoring'](#)
- [9]: [Alenezi et al. \(2020\): 'Harnessing deep learning algorithms to predict software refactoring'](#)
- [10]: [Golubev et al. \(2021\): 'One Thousand and One Stories: A Large-Scale Survey of Software Refactoring'](#)