UNIVERSITY OF MACEDONIA
GRADUATE PROGRAM
DEPARTMENT OF APPLIED INFORMATICS

PERSONALIZED DEVELOPER NAVIGATION IN SOFTWARE PROJECTS AIMING AT COMPREHENSIBILITY IMPROVEMENT AND FASTER PROJECT ONBOARDING

Master’s Thesis

of

Chrysochoidis Eleftherios

Thessaloniki, June 2022

PERSONALIZED DEVELOPER NAVIGATION IN SOFTWARE PROJECTS AIMING AT COMPREHENSIBILITY IMPROVEMENT AND FASTER PROJECT ONBOARDING

Chrysochoidis Eleftherios

Master's Thesis

submitted for partial fulfillment of

GRADUATE PROGRAM
DEPARTMENT OF APPLIED INFORMATICS

Supervising Professor
Chatzigeorgiou Alexandros

Approved by tripartite examining committee on 29/06/2022

Chatzigeorgiou Alexandros	Satratzemi Maria	Kaskalis Theodoros
....	....	....

Chrysochoidis Eleftherios

....

Abstract

Software projects nowadays are getting more sophisticated by utilizing the new technology and the increased hardware power to meet all client requirements. New tools and frameworks are keep coming up very frequently, generating many new opportunities and roles which are tempting, for the majority of the developers and this leads to high turnover rates. The turnover effect, the intensive complexity and the huge variety of new tools are combined the main reasons for which the comprehensibility deteriorates.

Low levels of comprehensibility imply more time for a new joiner to become productive which in turn is translated to cost. Due to high turnover rates such costs are massive and apart from delays and costs, many projects even fail.

Comprehensibility however, is not dependent only to the project itself. Soft Skills are very important in order to be able to grasp a project faster. Comprehensibility has been studied a lot as the issue was identified many years ago and it affects not only the Corporate section, but also the open source community, as contributing require a lot of time as there is none you could directly talk to help you onboard to the project.

A survey was conducted among 81 developers and students in order to spot code reading challenges and collect any best practices.

Combining the survey results with the literature review, a plugin was developed in order to assist developers to onboard to new projects as the clearest result from the Survey, was that everyone prefers the one-to-one direct meeting to onboard them on a project and CodeTour plugin that was developed in that context, could enhance this onboarding experience by providing an interactive way to onboard, without relying on anyone else.

Keywords: comprehensibility, turnover, intellij-plugin

Acknowledgements

A lot of effort has been put into getting this Thesis done, and I am more than proud that I have completed it. The topic was really exciting and it is present on real-world projects as well, so diving into it, helped me a lot to grow professionally.

The fact that this study along with the generated plugin was accepted to be presented in a huge global Developers conference, was one of the highlights of my so far career, and it proves the significance of the topic.

I would like to express my gratitude to my supervisor professor Chatzigeorgiou Alexandro, for all the guidance and inspiration he offered me. I feel very lucky and honored that I had the chance to work and collaborate with him, and I couldn’t have asked for more.

In addition, I would like to thank some of my colleagues and friends for the ideas and the feedback they gave me.

Last but not least, I wish to acknowledge a special person who is in my life for more than 8 years, my partner Konstantina, for all those times that she was there for me, listening carefully to my ideas and thoughts and without understanding nothing at all, she always inspired me for more.

Table of Contents

List of Figures

List of Tables

List of Code Blocks

1 Introduction

1.1 The Issue

Technology is rapidly evolving and in combination with higher computational power and faster internet speeds, lead to more sophisticated software projects which inevitably are more complex. **Complexity** is strictly related to **Comprehensibility** for a project. Typically, the more complex a project is, the less comprehensible it turns out to be. However, a software should be comprehensible because change and evolution are critical characteristics of a project's life. A software that doesn't evolve will be deprecated sooner or later as per first law of Software Evolution (Lehman, 1996).

In addition to Complexity, high **Turnover** rate is nowadays an issue which deteriorates the comprehensibility of a software project because when development teams are changing, there is some knowledge loss especially when important members leave (Nassif and Robillard, 2017). There are many reasons for which turnover rates have been increased, but turnover should not be considered as a root cause. Instead, its consequences are the ones which aggravate a project's comprehensibility.

A third cause could be considered the wide variety of **New Tools**, including new technologies, frameworks, languages and libraries, that are coming up very frequently which may affect comprehensibility in two ways: a) wide variety of tools means less expertise on specific tools and thus less strong candidates, b) older tools are rarely preferred by younger developers but sometimes they are asked to work with them and in such cases they struggle more.

Project Comprehensibility is an issue for which the academic community related to Software Engineering, has performed many studies throughout the years and it is still open to further research. Additional to academia, this topic also concerns the Open-Source community as well as the Enterprise world because in both cases there is an important need for projects to be comprehensible enough, so that development teams could deal with the requirements within a reasonable timeframe.

The importance of this issue may also be perceived from another point of view: Software having low levels of comprehensibility eventually lead to **delays**, **budget exceedance** and sometimes even to project **failures**.

1.2 Goals - Research Questions

The overall goal of this Thesis is to create a Tool aiming at comprehensibility improvement and faster project onboarding.

To accomplish that goal, a literature review and a market investigation is required, in order to find out some techniques and tools that have already been studied and evaluated for their efficiency. This can be summed up to the first Research Question (R.Q.):

R.Q.1: Are there any available methods, tools or patterns aiming at comprehensibility improvement?

Another interesting field of study is the factors that may affect the comprehensibility of project and what can developers do to avoid them. This can be summed up to the second Research Question (R.Q.):

R.Q.2: What affects a project's comprehensibility and how can it be maintained throughout all the development process?

After answering the two research questions, the development of a Tool aiming at comprehensibility improvement and faster project onboarding, will become the ultimate goal and the Thesis deliverable. The tool should be developed as a Plugin on a professional IDE in order to be used both by Students and Professional Developers and also a setup of a repository fully compliant with open source community should be delivered. This can be summed up to the following Research Objective (R.O.)

R.O.1: Tool development aiming at comprehensibility improvement and faster project onboarding.

1.3 Outline

The related research along with the findings and the development related work are the content of the remaining document. More specifically, Section 2 contains an overview of the related literature review, in Section 3 the Methodology following to accomplish Thesis' tasks is described, in Section 4 the survey results are presented, in Section 5 the development related tasks for CodeTour Plugin are provided and in Section 6 a demonstration for CodeTour's usage is presented, and conclusions are in Section 7.

2 Literature Review

2.1 Turnover And Effects in Productivity

The main reason for which the need for comprehensibility is now more important than ever, is due to the high Turnover ratio that is a fact in almost any company. Turnover defines the rate at which employees leave (either willingly or not) a company and are replaced by new ones. Such replacements have a substantial cost on companies due to the turnover effects and it is applicable on almost any domain. However, in Software domain it seems to be more crucial as ultimately the cost, the schedule and the efficiency of a software project are directly affected by high turnover ratio which eventually may lead even to project failures.

Turnover is widely studied across many domains and from different aspects (Gan and Zhang, 2010). There may be many and various reasons for high turnover rates and many studies have been devoted to find ways to lower those rates, but especially in Software turnover seems inevitable.

For this reason, researchers tried to define ways to measure those effects. A very elegant approach has been given by (Muhammad KHAN *et al.*, 2015) which tried to provide a framework able to calculate the Productivity Rate of software team taking into consideration some very important factors. The idea of the framework is to define the **Optimal Productivity**, which would be the productivity rate if nothing changed within the team, and calibrate it with the **Turnover**, in order to find the **Actual Productivity Rate**. The Turnover in that framework, consists of 3 basic factors:

a) The *Turnover Rate* which is the typical rate as defined earlier
b) The *Job Matching* which is an indicator which shows how much a new employee would fit to a position compared to the previous employee who was on that position. This indicator could be either negative (when the new employee is not as good as the previous one) or positive (when the new one turns out to be more a better fit)
c) The *Firm Specific Human Capital* (FSHC) which in simple words is the skills and the knowledge of an employee, that have productive value in one particular company

The described framework is perfectly depicted on KHAN's paper (Muhammad KHAN *et al.*, 2015):

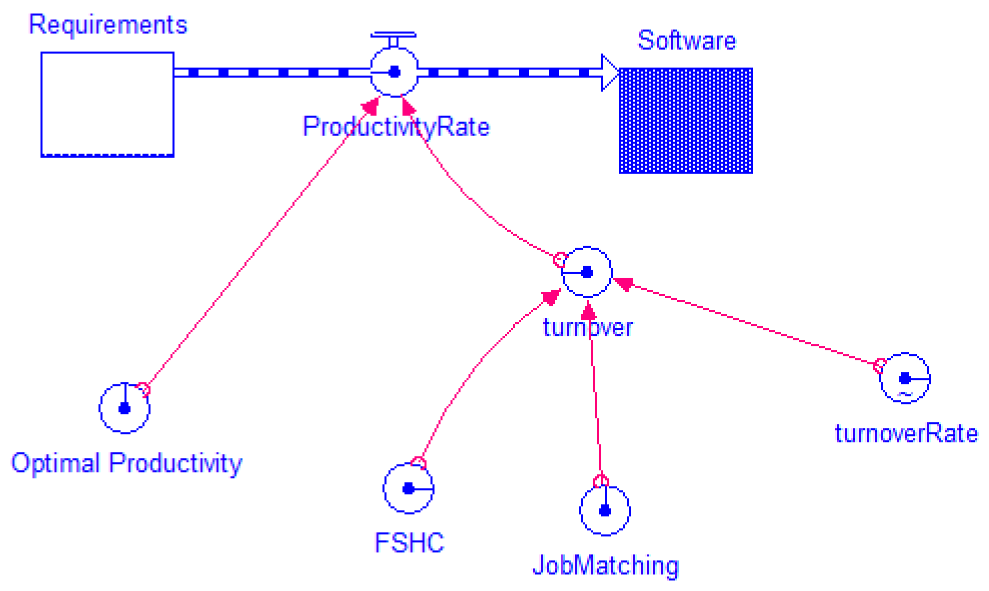


Figure 1. Productivity Turnover Modeling

KHAN also provided the mathematic formulas which can calculate the relation of Optimal and Actual Productivity depicting them into diagrams for better visual comparison. The formulas and the diagrams are suggested to be used for what-if scenarios in terms of evaluating the current situation and trying to predict the future changes as well.

The knowledge of such relations, may lead to important decisions to be taken a-priori to prevent dealing with unfavourable cases.

2.2 Comprehensibility Value

From a project-first perspective, Code Reading is not easy and becomes more and more challenging and often there are cases where even the authors struggle to deal with their own projects (Valentino Vranić, et al, 2015). Sometimes this may be the result of the Technical Dept that is accumulated on a project throughout the years and is never paid back, while in other cases the lack of documentation or its poor quality may lead to such results.

Many documentation techniques have been studied and evaluated in the past (Garousi *et al.*, 2013) and in most of the experiments the results always shown that a good quality documentation is valuable (Plosch, Dautovic and Saft, 2014). Documentation quality and formats is a topic that concerns the Software Engineering academia for many years as the value was clearly visible and the need for it was about to become more and more important as correctly predicted (Curtis *et al.*, 1989).

The reasons that maintaining a documentation is so challenging can be grouped on 2 dimensions: a) the Extra artifact and b) the lack of Measurement framework.

Documentation sometimes is part of the code either able to be extracted or not, and other times is just an externally maintained project in any format. In both cases it is considered an Extra artifact, as its purpose is only internal and it gives no value to the client or the end user. Thus, extra effort is required in order to maintain it, and this effort most of the times is not included on project estimations and just because clients don't care about it, documentation is the first task that is always omitted when there is not enough time (despite there is almost never enough time).

However, even when a team cares about their documentation and they dedicate some time to maintain their documentation, there is not any validation mechanism that can verify the quality of the documentation. In other words, documentation cannot be measured and thus, its quality is ambiguous. How can something be evaluated if there is not a way to measure it? Even if there is documentation for any section, any class and any line of the code doesn't mean that it is good quality documentation. Besides, code is just **text**, and documentation tries to explain this text to others. However, code is being tested as the *"text"* is going to be transformed into features which will be tested by clients and end-users, but for documentation that's not the case unfortunately (Valentino Vranić *et al.*, 2015).

Comprehensibility is not easy and a project would rarely be very easy to read. This is an assumption made by many researchers and professionals, and because reading or joining a new project is a very common case for a developer, its significance has been acknowledged. Being a developer is not easy (Spinellis, 2018) as reading and understanding existing code in order to extend it to meet new requirements and features, or fix issues and bugs or refactor it to reduce technical debt are tasks that a developer will be requested to do. In addition, a developer should be able to grasp the detailed logic of a code as much as possible, either they are the authors or the code is written by others, to summarize the code into concise comments and share that knowledge to their colleagues. This is of course part of the collaboration skills as well.

2.3 Software Comprehensibility Factors and Tools

The challenge for a project's Comprehensibility lies on many factors. One of them is proven to be the documentation as mentioned earlier and another factor is the quality of the project itself. Another factor is the cognitive skills that the developer who is going to read/onboard to the project may have.

2.3.1 Cognitive Skills

Cognitive skills are under the umbrella of Soft Skills and are considered very valuable and complement to Hards Skills as for example Java knowledge (Schulz, 2008). Although Cognitive Skills are not directly related to a project's Comprehensibility, they may play a significant role to an onboarding process as each new joiner may have different mindset, which could make the onboarding experience vary (Valentino Vranić *et al.*, 2015). This conclusion led to many researchers focus on studying comprehensibility from this point of view, having in mind that at the end a project's comprehensibility depends on the skills that the new joiner may have.

2.3.2 Spatial Skills

Going one step further, (Jones and Burnett, 2007) limited the Cognitive Skills that affect positively an onboarding process to the Spatial skills. Spatial skills are defined as *"the ability to generate, retain, retrieve, and transform well-structured visual images"* (Lohman 1996). A developer having strong spatial skills, is more likely to onboard faster and more efficiently on a new project compared to others as they will be able to extract key information easier and navigate to the code following a more structured path. In addition, they also state that before diving into the onboarding process and code navigation hard skills like the level of skill in the project's language does matter a lot.

2.3.3 Contributing to Open Source

The importance of cognitive skills has already been recognized by several Professors and as a result many of them have adopted it to Software Engineering courses where the students as part of the evaluation process are being asked to contribute to an open source project. Many controlled experiments have been conducted and their results are very satisfying (Pinto *et al.*, 2019) as students exposed to such tasks, seem to have gained some significant benefits. To further support and spread this method, some researchers also studied the best practices and methodologies for finding suitable open source software projects in order to teach Software Engineering (Smith *et al.*, 2014), taking into consideration the programming language, the size of the code base, the projects activity (if it is active or not) and they even provide ways to search for such repositories! Teaching Software Engineering through engagement with open source projects is a win-win situation and the more a student is involved, the more ready would become for the real world projects.

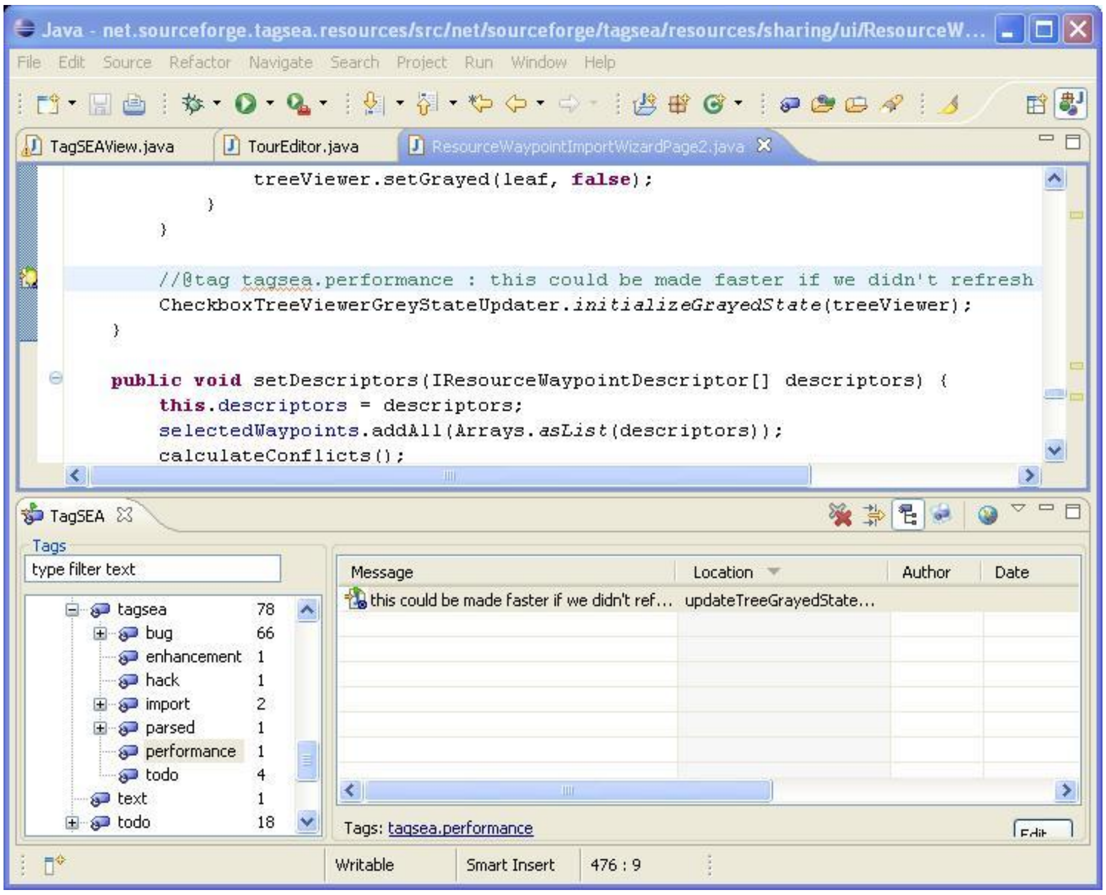
2.3.4 UML

Although cognitive skills are important, there is still the need to make a project as friendly as possible for new joiners. To enhance the so-called onboarding process, Software Engineering researchers came up with tools such the UML (Unified Modeling Language). Cases studies has shown that UML can improve the comprehensibility of a project as well as the traceability on the code (Anda *et al.*, 2006) but more recent research shows that UML is slowly abandoned and especially on open source projects where the usage of UML is very discouraging as it is only 0.28% (less than 1%!) according to Hebig et al., 2016. The main reason seems to be that nowadays most projects are web applications and they don't make use of Object Orienting Programming (OOP) features because they are not needed. UML can be very useful to describe a structure having classes, interfaces, inheritance as their relations could be easily identified by just viewing the UML diagram. However, in modern web applications languages and frameworks based on JavaScript are being widely used for which UML diagrams wouldn't add any value. In addition, in world where microservices are almost everywhere, having a Model View Controller (MVC) pattern, makes the use of UML unnecessary, as the requirements for an architecture including microservices are focused on visualizing the communication between them and other components.

The only domain where UML could still provide some added value is on projects where there is a complex business logic implemented using OOP features. There are many enterprise applications which are still using such approaches and UML could still be used there. However, even for such cases, maintaining the UML diagrams during code changes is a big challenge (Fernández-Sáez, Genero and Chaudron, 2013). UML diagrams should ideally be synchronized with the code and some approaches have been suggested to automatically sync UML diagrams and the source code (Cazzola, Ghoneim and Saake, 2006) but they haven’t managed to be established to the community.

2.3.5 *TagSEA*

In 2006, an interesting tool has been developed and its usage was evaluated having interesting results (Storey *et al.*, 2007). The tool was called TagSEA, which stands for Tagging of Software Engineering Activities, and it let developers turn their comments into navigatable waypoints. The tool was available as a plugin to Eclipse IDE and its concept was very similar to CodeTour’ s one: Tagging code parts in the code, adding meta information for the related code block and showing all the available tags (with grouping feature) with ability to navigate on their location and see their meta information. An example can be seen in the figure below:


Java - net.sourceforge.tagsea.resources/src/net/sourceforge/tagsea/resources/sharing/ui/ResourceW...
File Edit Source Refactor Navigate Search Project Run Window Help
TagSEAView.java
TourEditor.java
ResourceWaypointImportWizardPage2.java
TagSEA
Tags
type filter text
Writable
Smart Insert
476 : 9

```
                treeViewer.setGrayed(leaf, false);
            }
        }

        //@tag tagsea.performance : this could be made faster if we didn't refresh
        CheckboxTreeViewerGreyStateUpdater.initializeGrayedState(treeViewer);
    }

    public void setDescriptors(IResourceWaypointDescriptor[] descriptors) {
        this.descriptors = descriptors;
        selectedWaypoints.addAll(Arrays.asList(descriptors));
        calculateConflicts();
```

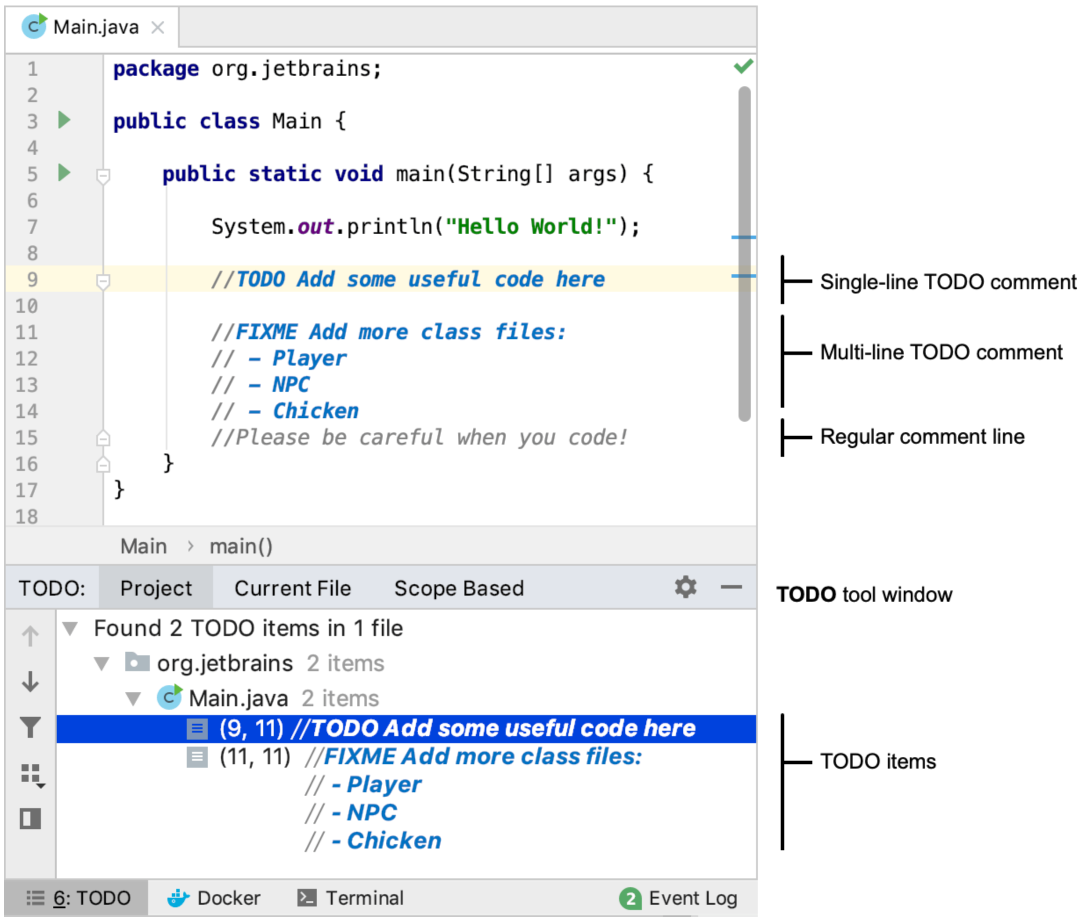
tagsea	78
bug	66
enhancement	1
hack	1
import	2
parsed	1
performance	1
todo	4
text	1
todo	18

Message	Location	Author	Date
this could be made faster if we didn't ref...	updateTreeGrayedState...		

Tags: tagsea.performance

Figure 2. TagSEA Eclipse Plugin

Although the idea of TagSEA was innovative, it didn’t get much publicity and as a result it is not active anymore. However, some parts of its functionality have been adopted by many IDEs, as for example the TODO tasks collection. On IntelliJ-based IDEs, comments that include the TODO or FIXME placeholders, are automatically being collected and shown on a tree, on the TODO tool pane. An example is shown below.



```
package org.jetbrains;

public class Main {

    public static void main(String[] args) {

        System.out.println("Hello World!");

        //TODO Add some useful code here

        //FIXME Add more class files:
        // - Player
        // - NPC
        // - Chicken
        //Please be careful when you code!
    }
}
```

```
Found 2 TODO items in 1 file
  org.jetbrains 2 items
    Main.java 2 items
      (9, 11) //TODO Add some useful code here
      (11, 11) //FIXME Add more class files:
               // - Player
               // - NPC
               // - Chicken
```

Figure 3. IntelliJ TODO Tool Window

3 Methodology

Having completed the related literature review, the methodology that was set up in order to answer the Research Questions and accomplish the Research Objective, consisted of the following 2 parts:

a) Creation and sharing a Survey to collect feedback from professional Software Engineers and students

b) Development of CodeTour plugin for IntelliJ-based IDEs as an open source repository friendly to any willing contributor.

3.1 Survey Tool

For conducting a Survey, the most commonly used tool is Google Forms as it well-known, easy to use and supports all the basic requirements of a Survey.

However, for my case the option of Google Forms was inadequate due to the lack of responsiveness and the limited design options. Having a Survey with responsive design makes it much more accessible and as a result the potential target audience is increased. Thus, I did some research and after evaluating some of the available options, I decided to use SurveySparrow for my Survey.

SurveySparrow provides some very cool features, including custom survey domain, advanced question types, animations and step by step progress and of a very nice and responsive design that made the Survey looks very well both in browsers and mobile screens. Of course, features like email notifications, *"send me a copy of my response"* option, dashboard with analytics and charts and data export are supported.

A dedicated domain automatically created for the purpose of my Survey through which I could create questions, design survey's flow, share it and check results as well.

- The URL for my domain is: https://codetour.surveysparrow.com/ *(requires login as it is the management page)*
- The link to the survey is: https://codetour.surveysparrow.com/s/code-reading-challenges--best-practices/tt-080a698c44 *(shared link for Survey)*

3.2 Survey Questions

The Questions that were selected to be included on the survey were evaluated taking into consideration whether their responses could potentially:

1) Answer a part of a Research Question, or provide some useful context
2) Indicate patterns or best practices followed by professional developers and/or students
3) Evaluate the idea of creating a tool such as CodeTour (early feedback).

After the evaluation, the final questions have been collected and grouped into 3 sections:

1) **Demographic:** To be used for Statistical Analysis of survey's answers, with the ambition of ending up to meaningful results. Estimated time: <1 minute
2) **Project Info**: Users are prompted to select a project to have in their mind for which you want to provide feedback for. The project could be either their current or any past project. All the questions of this section were related to that project. Estimated time: ~1 minute
3) **Onboarding and Knowledge Transfer**: Onboarding on new project is challenging both from new joiner's point of view and from instructor's one. The questions of this section, aim to identify habits and trends from both aspects. Estimated time: ~7 minutes

The full Survey exported on a PDF file is available and can be accessed from here Survey - Code Reading Challenges & Best Practices.pdf.

In addition, results (the responses) from this survey can be accessed from here Results Report - Code Reading Challenges & Best Practices.pdf

3.3 Platform Investigation

For the implementation of the tool, a quick investigation of the available options performed, in order to decide the IDE for which the plugin would be implemented for. The following 3 options evaluated, and the highlights of their evaluation are depicted on the following table (✓=pros ✗ =cons):

Table 1. IDE Candidates Evaluation

IDE	Evaluation
IntelliJ	✓ Open Source
	✓ Documentation
	✓ Widely used by professionals (including myself)

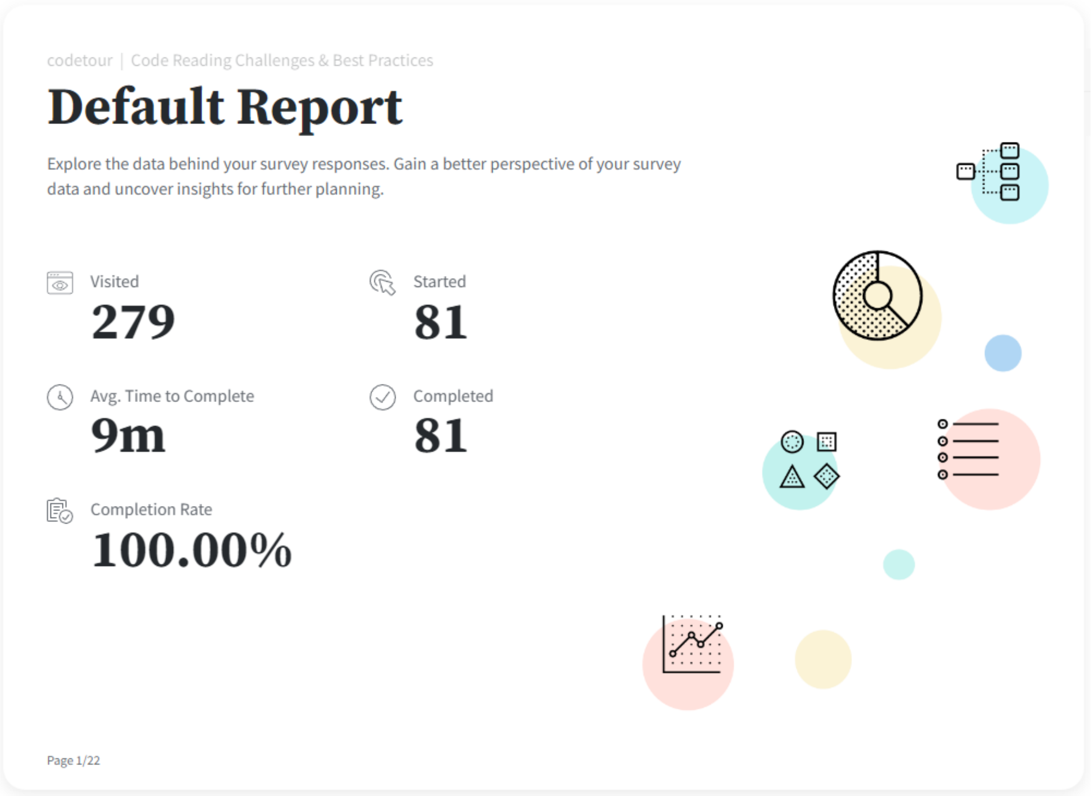
	✓ Multiple IDEs (PyCharm, WebStorm, CLion, GoLand etc)
VS Code	✓ Open Source
	✓ Documentation
	✓ Widely used by professionals
	— Familiarity (personal usage)
	✗ Existence of similar extension
Eclipse	✓ Open Source
	✗ Documentation
	✗ Rarely used by professionals nowadays

The prevailing IDE turned out to be **IntelliJ**, developed by JetBrains which is the main contributor and community orchestrator for the open source IntelliJ Community edition IDE (Github source code repository).

In case there was no similar extension, VS Code would probably be selected, because it provides the great feature of being able to run as an embedded web code editor. In simple words that means that VS Code can be launched on any modern browser having the preferred settings and extensions of a developer (requires login) making it a portable editor with no need for any installation.

4 Survey Results Analysis

The Survey was promoted to many different channels, including LinkedIn, Facebook, Reddit, direct email on Software Companies and University students. In total 81 answers have been received for the Survey. The dashboard's home screen of provided a summary of the responses:



codetour | Code Reading Challenges & Best Practices

Default Report

Explore the data behind your survey responses. Gain a better perspective of your survey data and uncover insights for further planning.

Visited
279

Started
81

Avg. Time to Complete
9m

Completed
81

Completion Rate
100.00%

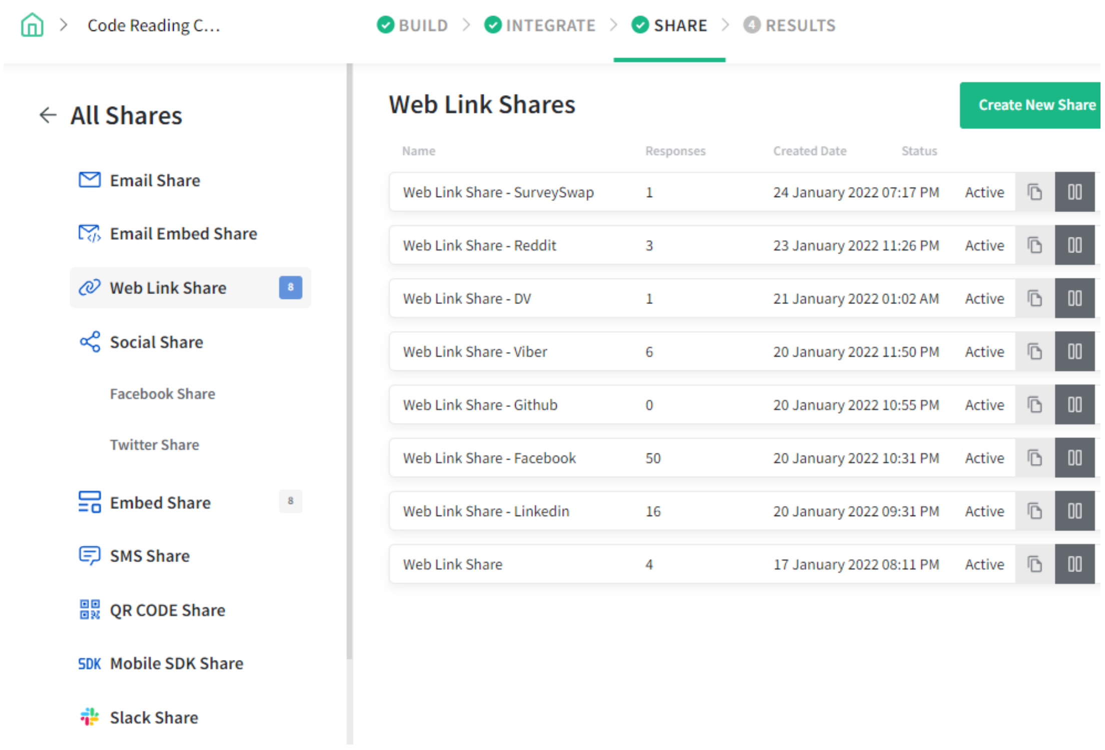
Page 1/22

Figure 4. Survey Summary

The length of the survey was quite long, as the average time for the 81 respondents needed to complete was 9 minutes. However, every single respondent that started the survey, successfully completed it (100% completion rate)! This high rate is probably due to the warning message that the Survey provided to respondents before starting, defining the target audience, and prompted them to continue only if they are included on it.

4.1 Source of Responses

The various responses came from different channels, but in majority they source was social media links. This was tracked using Survey Sparrow's feature for shared links which provides the ability to create different links to the same Survey but with some meta-data that they simply register the source name as defined on shared link creation. The shared link dashboard for CodeTour' s Survey (including their responses) is the following:


Code Reading C…

BUILD > INTEGRATE > SHARE > RESULTS

All Shares

- Email Share
- Email Embed Share
- Web Link Share 8
- Social Share
  - Facebook Share
  - Twitter Share
- Embed Share 8
- SMS Share
- QR CODE Share
- Mobile SDK Share
- Slack Share

Web Link Shares

Create New Share

Name	Responses	Created Date	Status
Web Link Share - SurveySwap	1	24 January 2022 07:17 PM	Active
Web Link Share - Reddit	3	23 January 2022 11:26 PM	Active
Web Link Share - DV	1	21 January 2022 01:02 AM	Active
Web Link Share - Viber	6	20 January 2022 11:50 PM	Active
Web Link Share - Github	0	20 January 2022 10:55 PM	Active
Web Link Share - Facebook	50	20 January 2022 10:31 PM	Active
Web Link Share - Linkedin	16	20 January 2022 09:31 PM	Active
Web Link Share	4	17 January 2022 08:11 PM	Active

Figure 5. Survey Response Sources

Note: The *Facebook* link includes the responses that came from direct emails to some students as the same link used on the email.

Another interesting meta-info that SurveySparrow provides, is the device source. As the below diagram indicates, the responses were almost equally distributed into mobile (41/51%) and computer (39/48%) while there was just 1 answer from a tablet. The high number of mobile responses, indicates the importance of having a responsive Survey, as it seems that many people prefer to use their mobile phones to answer surveys and especially if they find the survey on a social media platform.

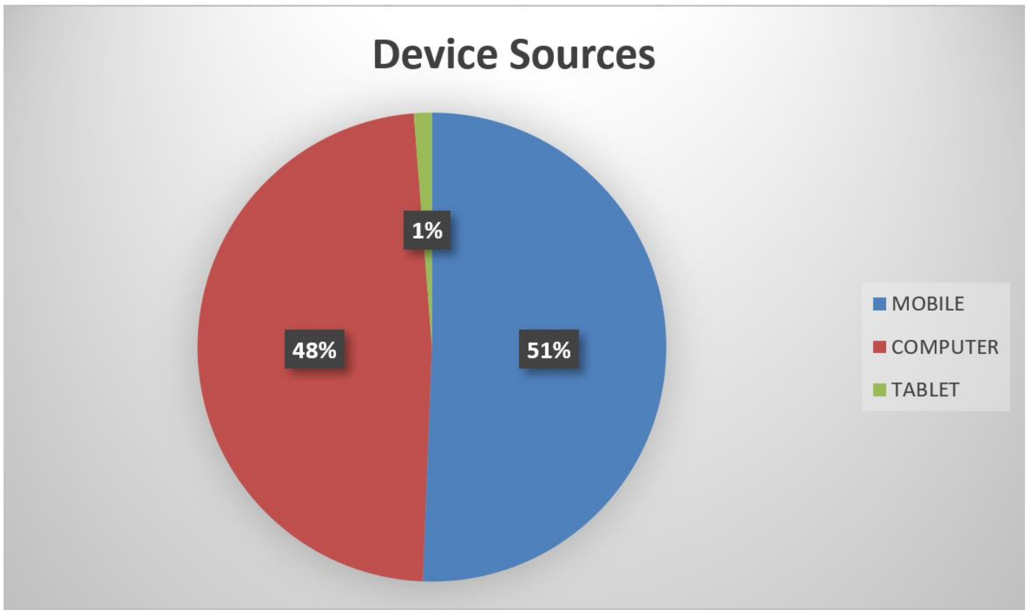


Figure 6. Survey Device Sources

4.2 UML Usage

Regarding UML and its usage, results show that UML is indeed not used that much nowadays and also it is not the preferred way for reading a project. This can be verified by 3 related questions: a) the direct UML usage question (question 14), b) the tool/methodology assistance evaluation (question 15) and c) the open source contributing pattern (question 16) for which the available options "*UML diagrams - for better understanding the designed structure*" and *"Read UML diagrams - to check where your feature could be added"* respectively were both ranked last.

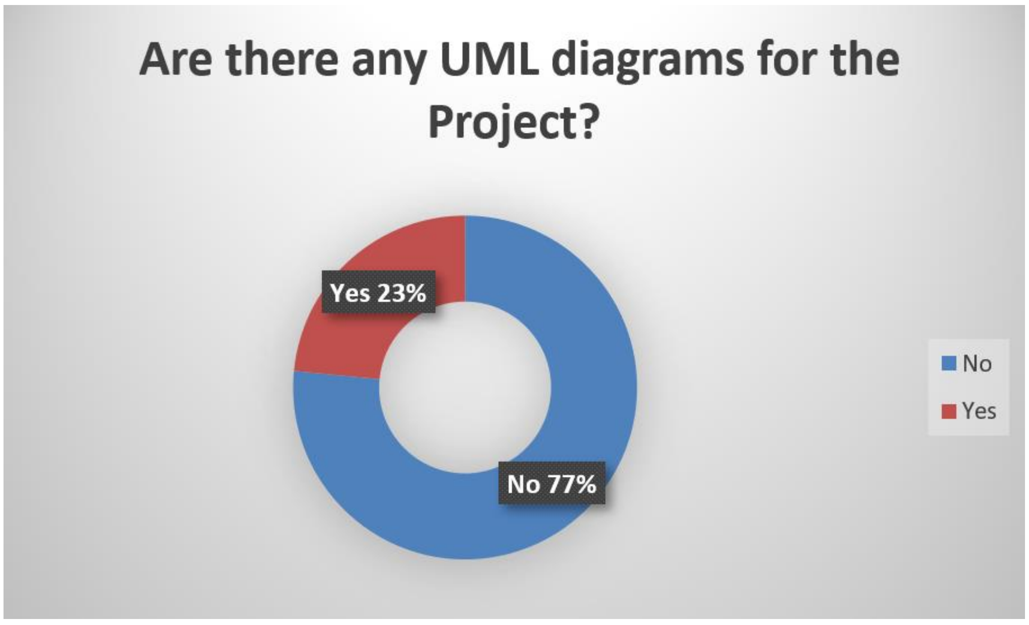


Figure 7. Survey UML Usage

4.3 Patterns and Methodologies for Code Reading

In order to extract some useful results related to Code Reading the responses to following 2 Survey questions were analyzed:

- *Consider that you came up with a cool feature on an Open Source project, and you want to contribute on it, to implement it. However, you don't have anyone to provide you information directly, and you have limited time, so you need to do your contribution without spending too much time. Which of the following actions, would you prefer doing, in order to get the "quick win"?* (Question 16, 4 options)
- *Consider that you are responsible for the 1st week of training of a new member on your Project (the project you selected before). Which of the following, would you provide to your colleague, for that early stage? Please select only those that are available to your Company and your selected Project.* (Question 17, 6 options)

For fast contributions to open source projects, no clear preference or trend was identified, as there seems to be a balance between the given options.

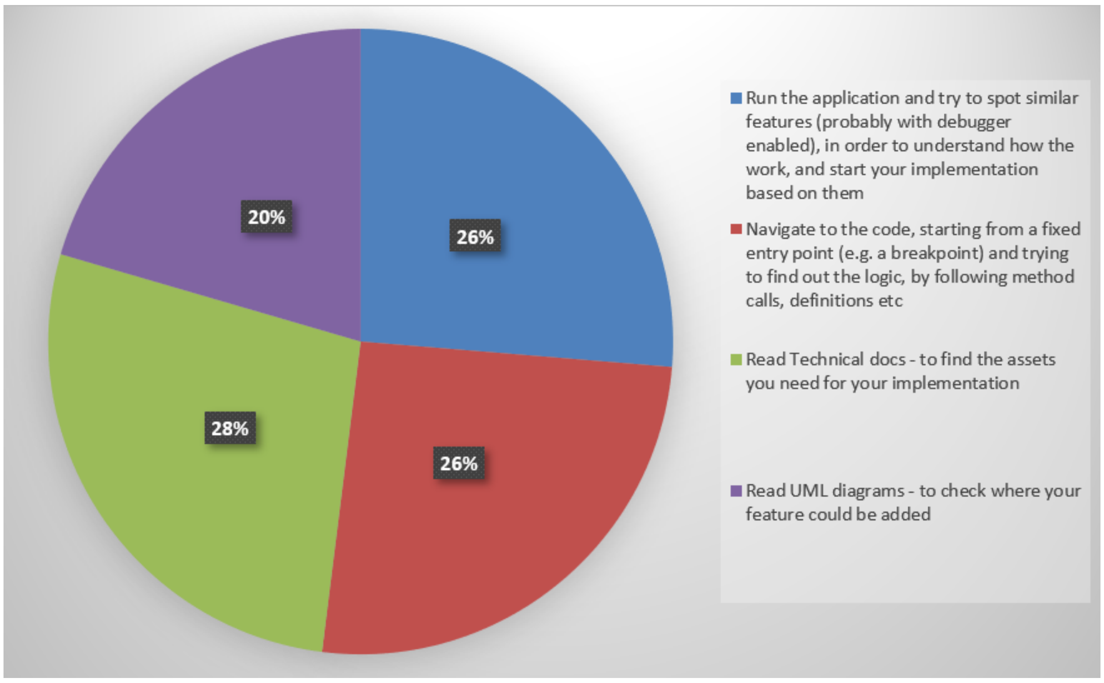


Figure 8. Survey Open Source Contribution Preferences

When it comes to onboarding, there seems to be a clear preference on One-to-One trainings where project is demonstrated and code is explained in detail.

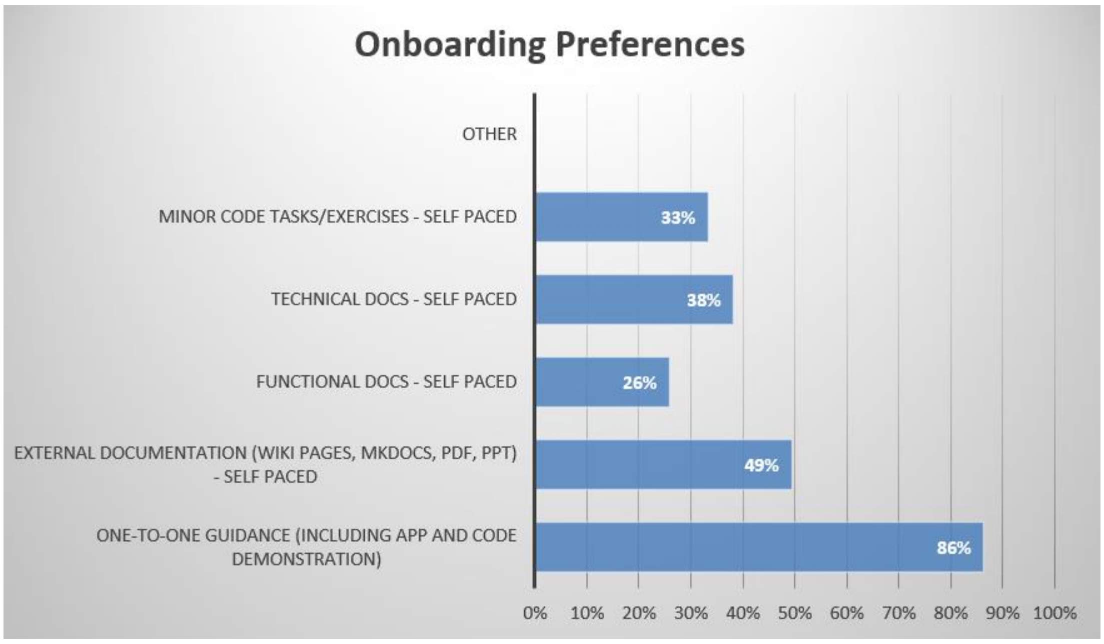


Figure 9. Survey Onboarding Preferences

An interesting challenge here is to find the relation between the time needed for One-to-One compared to the time needed for maintaining documentation and training material (all the other options) because that time might eventually be more than the One-to-One would need.

4.4 Comprehensibility Factors

To identify factors that could affect Comprehensibility, respondents were asked to provide their rating for any of the available options (Question 15, 12 options). If a project uses good and consistent naming conventions for classes, packages, variables etc and in general it is well structured, that is considered a good factor and in conjunction with good quality of documentation (both inline and external documentation) makes a project very comprehensible.

Another interesting result is the confirmation of the concept that the level of a developer affects the time they need to onboard. That means, that the more senior level the developer is, the less time they will need to onboard as shown below.

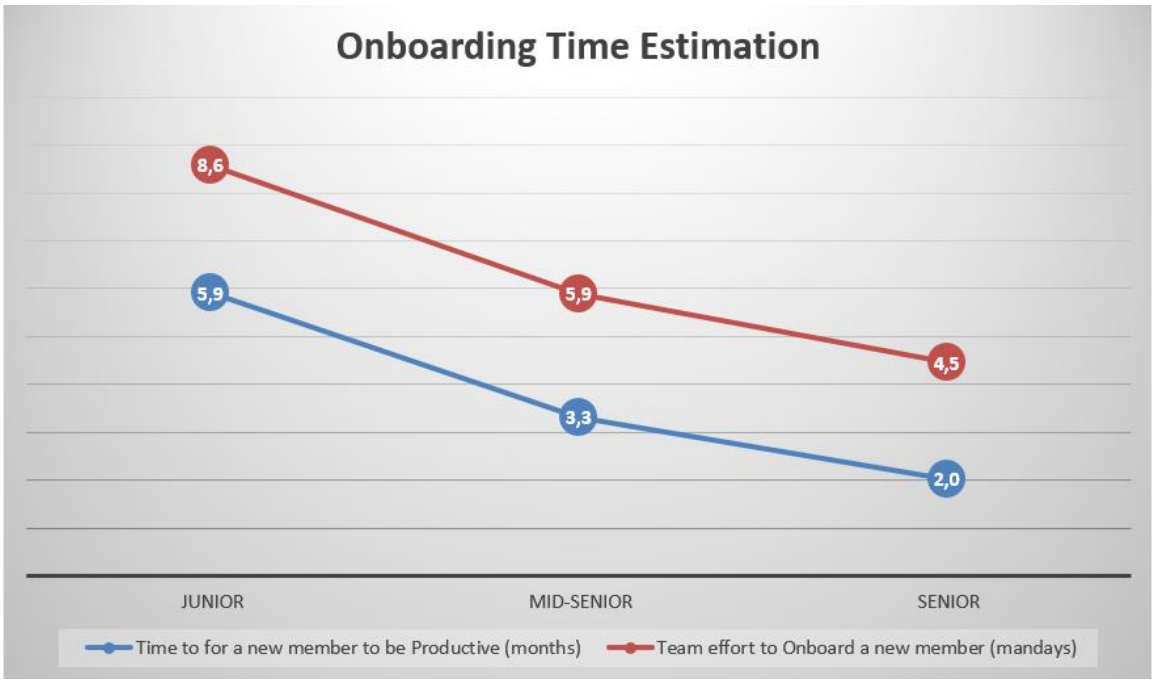


Figure 10. Survey Seniority Level and Onboarding Time

4.5 CodeTour Early Feedback

Some early feedback has been received for CodeTour both directly and indirectly. Initially, the responses on the favorite IDE question in which IntelliJ and VS Code are clearly more famous compared to the others with IntelliJ being first with small difference. Thus, CodeTour's target IDE could be IntelliJ.

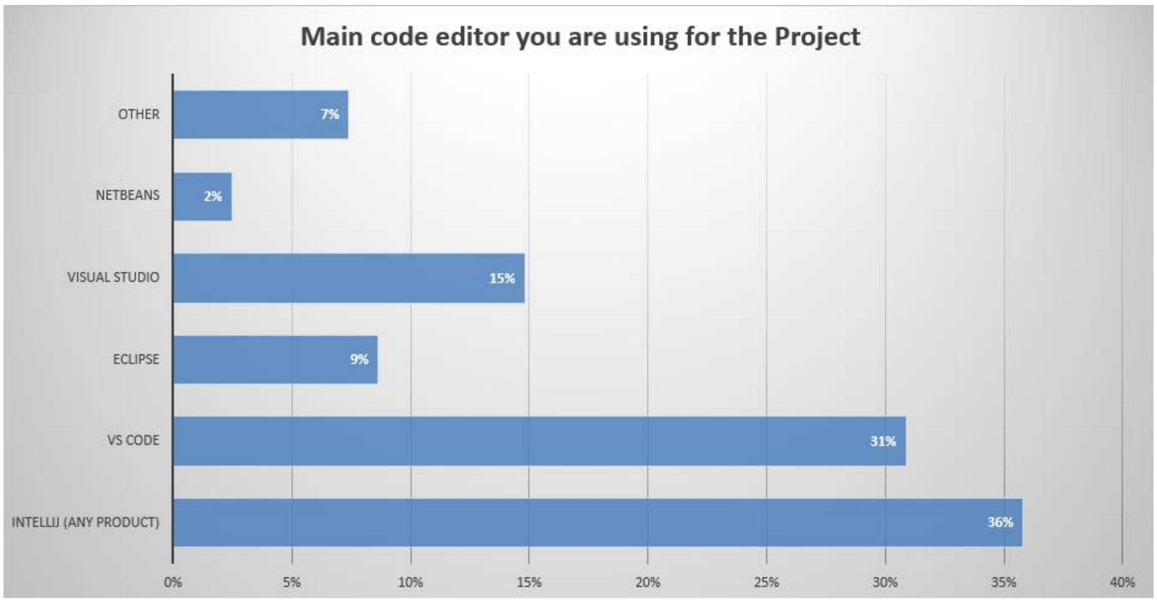


Figure 11. Survey Favorite IDE

For the direct part, respondents were asked to evaluate how useful would a tool such as CodeTour could be, by providing them the description of the tool and its features. The question was Rate questions (0-5). Having an average of **4.1/5 (82% agree)** CodeTour sounds good as an idea and thus implementing such a tool, could probably worth it.

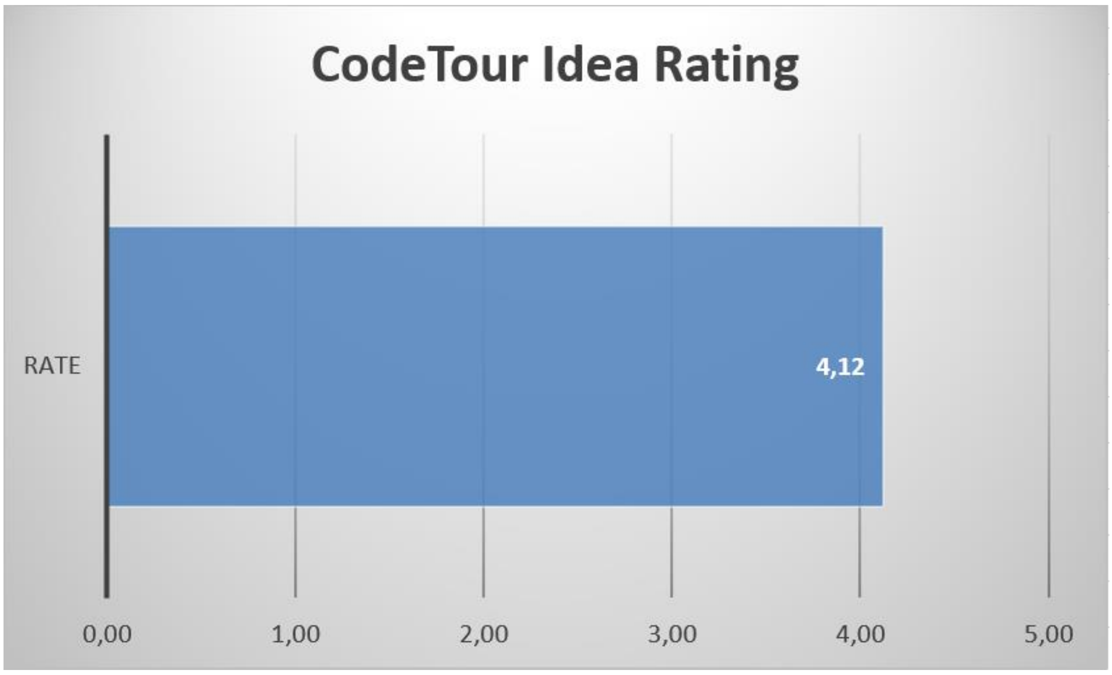


Figure 12. Survey CodeTour Idea Rating

5 Code Tour – Plugin

5.1 Introduction

CodeTour is a plugin, that provides the ability to record and play back guided walkthroughs of a project’s codebase. It's like a table of contents, that can make it easier to onboard (or re-board) to a new project/feature area, visualize bug reports, or understand the context of a code review/PR change. A Code Tour is simply a series of interactive Steps, each of which is associated with a specific file/line, and includes a description of the respective code. This allows developers to clone a repo, and then immediately start learning it, without needing to rely on others for direct assistance. Tours can be version controlled into a repo, to enable sharing with other contributors.

5.2 Origin

The idea of such a tool originally came to my mind, when I was member of the Development team of a startup company, in which we used to build a whole new Software Product from scratch. In the beginning everything was fine, as we were 7 Developers dedicated to our project but after a while, our company started to grow and we needed more and more stuff. After hiring a couple of developers, we noticed that the onboarding process was very time consuming for us and not so effective for our new hired colleagues. Our Project had grown so much, that it was very challenging even for us to explain it to new joiners, as there were many aspects of the project that were not easy to be fully grasped.

Every developer that was with us from company’s day 1, was confident enough for our project, as we were a small team with great communication and very good skills. Our challenge was to make the new joiners productive as soon as possible, and obviously, with the lowest possible effort, as requirements and due dates were kept coming.

Unfortunately, that was not the case for us. Each new joiner, was dependent to us for almost anything: Preparing their development environment, their local environment for running the application, building the application along with each of their component/service, and ultimately understanding the business logic and how it was implemented on our project. There were many times when we wondered what we could do in order to help our new colleagues be more confident and productive. Obviously, “*private lessons*” was not an option, as the working pace on Startup companies especially on early stages is often very high.

So, one day we had a conversation as a team, in order to evaluate our options for this challenge. Many options were mentioned as for example the Video Recordings or the extended Training Material but unfortunately, they couldn't fit to our case. The pros and cons of the evaluated options are briefly described on the following table.

Table 2. Evaluated Options for Onboarding

Evaluated Option	Pros	Cons
Video Recordings	- Record once, Playback multiple times - Explanation and Code Navigation - Self-Paced (for the new joiner) - Efficiency	- Hard to maintain due to code changes as editing only parts of a video - Detailed Explanation requires much time to generate (good preparation, multiple shots, editing)
Training Material - **Presentation Decks**	- Multiple levels (could be separated be service, per level etc) - Self-Paced	- Maintenance - Outside IDE - Efficiency (such documentation is good but not sufficient)
- **External Documentation**		

As our project was rapidly evolving, including new features, refactors, more components etc, we didn't have much time to properly prepare any of the available options. It was those days when I thought that this was a really big issue, because such cases:

- Slow down our development speed (as individuals)
- Slow down our overall progress (at least until new joiners will become productive and caught up with the "cost" we invested for our time). More communication, less development
- Pose a potential threat for business continuity in case development team was entirely replaced (eventually)

I thought it would be great if somehow, we were able to provide our new joiners a tool through which they can navigate to our project's code and having some explanation at the same time for the context of the specific part/logic. Thus, that was the main reason due to which I proposed the topic of the current Thesis.

I had great expectations for this plugin, and I was sure that developers will love the tool, and I was very happy that I will be the one who will create it! When I started my research, I also created the Github repository for the plugin, which I originally named it **CodeTrailer** as I thought it was catchy enough to stimulate developers' interest.

However, after some research I came across a great VS Code extension with the name CodeTour, authored by Microsoft. This extension was very similar to what I was envisioned and unfortunately there was someone who has built it before me. Well, that was a small disappointment to me, but I thought that there is still the opportunity to implement the same tool on JetBrains related IDEs. And so I did! I changed my plans, renamed Github repo to CodeTour, and focused only to IntelliJ platform.

Note: As there are actually 2 different tools with the same name, any reference to CodeTour might be confusing. Thus, to make it clear, from now and on:

- **CodeTour** indicates my CodeTour - IntelliJ plugin while
- **VS Code CodeTour** indicates the CodeTour – VS Code Extension

However, it is important to mention that CodeTour and VS Code CodeTour are not competitor tools. They are supposed to offer similar functionality but to different IDEs.

5.3 Features

As mentioned, CodeTour is available for free for both VS Code and for all IntelliJ-based products and can be found:

- CodeTour: CodeTour - IntelliJ IDEs Plugin | Marketplace (jetbrains.com)
- VS Code CodeTour: CodeTour - Visual Studio Marketplace

CodeTour is currently in version v0.0.3 (always check the latest) in which the minimum required functionality has been implemented and the supported features are briefly the following:

- **Tree-like View** of Tours and their Steps on a Tool Window
- **Code Navigation** with single click (on Steps available on Tours Tree)
- **Shortcuts** for Navigating on Previous and Next Steps
- Easy new **Steps Creation** through Editor's Gutter context menu (similar to adding breakpoints)
- Enhanced **Step Editor**, supporting ***Markdown*** and ***HTML*** for description along with real time **Preview Mode**

- **Customizable** (location, size, font etc) **Popup Window** for Step's description
- **Version Controlled** Tours for sharing/maintaining them

Some extra features have been registered and will soon be implemented from which the most important are probably:

1. Nested Tours
2. Export Tour on independent navigatable file
3. Navigate on a Directory/Package in Project Structure View
4. Selecting code block as a Step (instead of just *file:line*)
5. Dynamically Step update, on code changes

Full list of Open Issues (including bugs and features) is available on Github repository https://github.com/LefterisXris/CodeTour/issues

The Repository is Open Source so extra features may be implemented based on Community's requests. Further information about CodeTour's repository can be found on the related section 5.4.2.

5.3.1 Features Comparison

VS Code CodeTour is already 2 years old and is obviously a more mature project with more than 50 versions so far. The initial version was released on 08/03/2020 and since then many features have been implemented.

Comparing to CodeTour's current state, VS Code CodeTour has some additional features some of which are very nice:

- <u>Code Block Selection</u>: Instead of simply navigating on a file:line, a Step can navigate and highlight a specific code block
- <u>Special Elements</u>: On a Step's description special elements can be added. They will be rendered as links and on click some functionality is provided, for example
    - Open File -- *opens a specific file*
    - Open Tour/Step -- *opens the linked Tour or Step*
    - Insert Code -- *inserts a code block on the current cursor location (very useful for tutorials)*

    - Shell Command -- *executes a shell command e.g.,* `mvn clean install`, `npm install` *or even a custom command!*
- Exporting Tours: exports a single tour file, that contains the steps along with some extra context (typically 5 lines before and 5 lines after Step's location) of the related code, so that anyone can see that tour, without having the whole code
- Maintenance Automation Watcher: Github Action and Azure Pipeline for checking whether any tour is invalid (i.e., targets an invalid file, line or code block) to ensure that version controlled tours are always maintained
- Customization: Configurable settings for extension's behaviour, like Show Markers enable/disable and setting a Custom Directory where all tours reside instead of the default *.tours* directory.

Most of these features, are going to be implemented sooner or later on CodeTour. Ideally, both tools should have similar features and also it would be nice if there is compatibility between them, meaning that Tours created on a project using VS Code could also work for the same project if IntelliJ-based IDE is used.

5.4 Implementation

CodeTour is built using IntelliJ Ultimate edition. Its source code is mainly written in Java (92%) and there are some small code parts written in Kotlin (8%).

Developing an IntelliJ plugin, requires to have the IntelliJ Platform SDK (open source) as base dependency (provided) and some specific structure common for all plugins. This basic structure, along with many tutorials and all the available documentation of IntelliJ Platform SDK is kindly provided by JetBrains on their official documentation site: https://plugins.jetbrains.com/docs/intellij/welcome.html. JetBrains team has made a very good job on providing material and guidance for anyone interested to develop their own plugin. As a result, almost everything that I needed it was already defined on their documentation.

Gradle is being used as build tool for CodeTour and also Gradle Kotlin DSL is configured by default, providing powerful scripting and automation over Gradle.

5.4.1 Under The Hood

The general concept of CodeTour's implementation is the following:

- A Tour represents a group of Steps in a specific order. Holds extra information for its title, description and filename.
- A Step carries information for the location to navigate formatted as filename:line (e.g., Main.java:15), the description to render as Step’s explanation and its title.
- All Tours are visible on CodeTour Tool window as a Tree with each Tour being one branch of the Tree and each Tour having multiple branches for their Steps.
- New Tours can be created directly from the Tree through its context menu (right click on Tree’s root)
- Steps can be easily added on a Tour, by simply using the Editor’s Gutter context menu (right click on the file:line where the Step should point to) which will spawn a dialog so that developer can input the title and the description of the Step.
- Developers can use the Tree to initiate the navigation, by expanding the Tour they want and just selecting the first (or any other) Step. As soon as a Step is selected, IDE immediately opens the related file, navigates the user to the appropriate line and renders the Step’s description on a separate popup Window.
- Each Tour is persisted on a separate file with **.tour** extension. This file includes all the Steps for that Tour. Its structure is based on JSON.
- The base directory under which all Tours are persisted, is the **.tours** directory under Project’s root (it is automatically be created if it doesn’t exist).

Further implementation details are provided below for the most important parts of CodeTour’s source code.

5.4.1.1 Plugin.xml

The file plugin.xml, is required for any plugin and it holds information related to the declaration of a Plugin including actions, icons, shortcuts and services. It is located under *resources/META-INF* directory and it is the most important file of any IntelliJ plugin. The basic structure of a plugin.xml follows as an example, while more information can be found on Plugin Configuration File Documentation.

Code Block 1. Plugin.xml Structure

```xml
<idea-plugin>
   <id>org.uom.lefterisxris.codetour</id>
   <name>CodeTour</name>
   <vendor>lefterisxris</vendor>

   <depends>com.intellij.modules.platform</depends>

   <extensions defaultExtensionNs="com.intellij">
      ...
   </extensions>

   <applicationListeners>
      ...
   </applicationListeners>

   <actions>
      ...
   </actions>

</idea-plugin>
```

5.4.1.2 Tour Definition

Tour is defined in class Tour.java under *domain* package, and has the following fields:

Table 3. Tour Definition

Field	**Description**
id	A random UUID for uniqueness
title	Tour's title, visible on the Tree
tourFile	The filename that this Tour will be persisted in
description	A description of the Tour
steps	List with all the Steps of the Tour in the given order

5.4.1.3 Step Definition

Step is defined in class Step.java under *domain* package, and has the following fields:

Table 4. Step Definition

Field	**Description**
title	Step's title, visible on the Tree
description	Step's description. Can be MARKDOWN, HTML or simple text
file	The file reference that the Step should open
line	The specific line to which the Step should Navigate the developer

Code Block 2. Tour Structure Example

```
{
  "id": "70abd0f5-3fb7-4309-a9bf-97afeb28aa9b",
  "tourFile": "sampleTour.tour",
  "title": "Sample Tour",
  "description": "A Sample Tour",
  "steps": [
    {
      "title": "A Step 1",
      "description": "# A Title\n\nThis is **Step 1**!!",
      "file": "Main.java",
      "line": 14
    },
    {
      "title": "A Step 2",
      "description": "# Another Title\n\nThis is **Step 2**!!",
      "file": "Main.java",
      "line": 28
    }
  ]
}
```

5.4.1.4 StateManager

Tours and their related Steps are persisted and managed through the class StateManager.java. In general, all CRUD operations (Create, Read, Update, Delete) for them are handled by StateManager. For Read and Write operations Google's GSON library is being used, as both Tour and Step classes are serialized as JSON formatted objects and stored in .tour files.

CRUD operations are always performed within a transaction, using an API provided by IntelliJ. For example, to write into a file, the operation should be registered within a WriteAction.runAndWait(()->{…}) block. Example of persisting a Tour into a file:

Code Block 3. Persisting a Tour

```
WriteAction.runAndWait(() -> {
   // Persist the file, using IntelliJ's virtualFiles
   final VirtualFile vfile = toursDir.createChildData(this, fileName);
   vfile.setBinaryContent(GSON.toJson(tour).getBytes(UTF_8));
   reloadState();
});
```

In addition, StateManager holds the information for the active Tour and the active Step, which in simple words shows the Step to which the user navigated last. This information is required for Previous/Next Step actions to be functional.

5.4.1.5 Tours Tree

To add a custom ToolWindow on IntelliJ Platform, the ToolWindowFactory interface should be implemented in order to register a factory which will create the ToolWindow. CodeTour's factory class is ToolPaneWindowFactory.java which is used as factory for the ToolPaneWindow.java class.

The ToolPaneWindow, uses the StateManager to load the available Tours, creates a component of Tree type (internal IntelliJ component) for which the TreeRenderer.java is registered as Cell renderer (i.e. responsible for how the Tree will rendering). Listeners are registered on the Tree in order to provide the single-click Navigation and also Context menu actions for Tours and Steps.

Finally, ToolPaneWindow subscribes to 2 messaging channels for listening on changes that require Tree re-render. Messaging API (MessageBus) is provided by IntelliJ and handled internally. An example follows and more info can be found here Messaging Infrastructure (jetbrains.com).

Code Block 4. Example of Messaging API – Subscribe

```
project.getMessageBus()
       .connect()
       .subscribe(TourUpdateNotifier.TOPIC, (tour) -> {
   stateManager.reloadState(); // get the latest updates from State
   createToursTee(project); // re-create the tree
   selectTourLastStep(tour); // select the updated tour
});
```

Code Block 5. Example of Messaging API – Publish

```
// Notify UI to re-render
project.getMessageBus()
       .syncPublisher(TourUpdateNotifier.TOPIC)
       .tourUpdated(activeTour.get());
```

5.4.1.6 Step Generator

The TourStepGeneratorAction.java is responsible for handling the Step creation taking advantage the Editor's Gutter menu. To implement the required functionality, the class should extend IntelliJ's AnAction interface, and also be registered as action on plugin.xml along with its desired place to let IntelliJ platform know where to add that action.

Whenever a developer uses that Action, CodeTour retrieves automatically the file and the line of the location they selected, by taking advantage of the VIRTUAL_FILE and the *EditorGutter.LOGICAL_LINE_AT_CURSOR* as dataId from the action event and registers it as the location reference for the Step. After that, a dialog pops up to the developer in order to input Step's information. The dialog is of type StepEditor which is described below.

Code Block 6. Getting File and Line from Editor Gutter

```
// Get the file and line in which user clicked
final int line = e.getDataContext()
                   .getData("EditorGutter.LOGICAL_LINE_AT_CURSOR");
final VirtualFile virtualFile = e.getDataContext()
                                  .getData(CommonDataKeys.VIRTUAL_FILE);
```

5.4.1.7 Step Editor

Step Editor.java is a custom implementation of DialogWrapper and its purpose is to provide a friendly UI to the developer to insert their Step. Step Editor's description field can support MARKDOWN, HTML and of course simple text. The description will be properly formatted when it will be shown to the Developer through the Step Renderer. However, Step Editor is equipped by a Preview tab in which the description is rendered exactly as in Step Renderer, so that the Developer can check how Step's description will look like when rendered.

5.4.1.8 Step Renderer

Similarly, StepRenderer.java is also a custom implementation of DialogWrapper, and its purpose is to popup a dialog, having rendered Step's description.

This dialog stores User Settings regarding the location and the size of the dialog window, to reuse it for all next steps, giving a smoother user experience this way. The internal class UserSettings is used for this purpose.

Step's description can be either MARKDOWN, HTML or simple text. In any case Step Renderer makes use of a MarkdownParser class, provided by IntelliJ platform, through which it transforms any MARKDOWN text to HTML and then HTML is properly rendered on the dialog component having the appropriate format.

In addition, StepRenderer dialog provides 2 extra buttons for easy navigation to Previous/Next Step, directly from the dialog.

5.4.1.9 Navigator

The Navigation to a specific *file:line* indicated by a Step can be triggered by 5 different ways in CodeTour:

1. By single click on a Step from Tours Tree
2. Using the Previous/Next buttons from Tours Tree
3. Using the Previous/Next buttons from StepRenderer dialog
4. Using the registered Actions in Tools menu (Tools > CodeTour) where there is Previous Step and Next Step actions
5. By using the registered keyboard shortcuts which are
    a. `Ctrl + Alt + Q` → Navigate to Previous Step
    b. `Ctrl + Alt + W` → Navigate to Next Step

All of these cases make use of the Navigator.java class (either directly or through Tours Tree messaging API) and its functionality is to:

1. Locate the target reference, indicated by the input Step's *file:line* property (uses IntelliJ FilenameIndex utility method)
2. Open the target file on the editor, requesting immediate focus on the target line
3. Fires up the StepRenderer in order to show Step's description on the popup dialog

Code Block 7. Navigator's Functionality

```
// Find the appropriate file
final VirtualFile vFile = FilenameIndex
                   .getVirtualFilesByName(step.getFile(), projectScope);
// Navigate requesting focus
new OpenFileDescriptor(project, vFile, step.getLine()).navigate(true);
// Show StepRenderer
StepRenderer.getInstance(step, project).show();
```

5.4.1.10 Icons

CodeTour uses a special pattern for Icons management, which is very handy for IntelliJ Plugins development. The icon files are placed under resources as usual, and there is an interface called CodeTourIcons.java under package icons in which all the icons are loaded using IntelliJ Platform's utility class IconLoader, and this way, the icons can be used in any place of the code as well as in plugin.xml file.

5.4.1.11 Notifications

There are some cases in which CodeTour needs to show a notification to the developer. For such cases, CodeTourNotifier.java class has been created and provides the API for CodeTour notifications hiding all the boilerplate code, making notifications easier and more consistent. All notifications are also logged for easier debugging in case it is needed. An example of the definition and the related usage follows:

Code Block 8. Notification API Example – Error Definition

```
public static void error(@Nullable Project project, String content) {
   NotificationGroupManager.getInstance()
         .getNotificationGroup("CodeTour-Notification")
         .createNotification(content, NotificationType.ERROR)
         .setIcon(CodeTourIcons.LOGO_S)
         .notify(project);
   LOG.error("CodeTourNotifier: " + content);
}
```

Code Block 9. Notification API Example – Error Usage

```
CodeTourNotifier.error(project, "Unexpected Tour Error");
```

In order to register a custom Notification group, there should be a declaration in a ***notificationGroup*** tag, giving the id of the group, the display type and the default icon.

Code Block 10. Registering a custom Notification Group in plugin.xml

```
<notificationGroup id="CodeTour-Notification"
                   displayType="BALLOON"
                   icon="CodeTourIcons.LOGO_S"/>
```

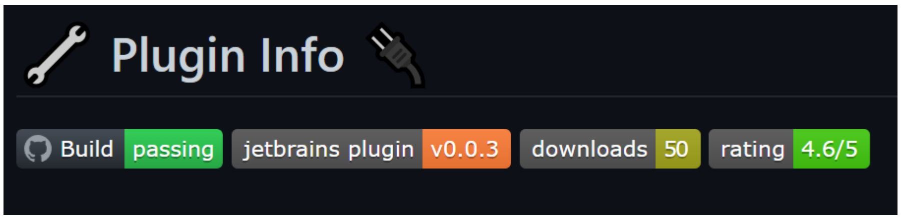
5.4.2 Github Repository

CodeTour is open source, and its code is publicly available on Github on the following link: https://github.com/LefterisXris/CodeTour under MIT license which in simple words means it is available for modification, distribution and even for commercial use.

Repository's README.md contains a very brief explanation about the project, its structure, useful metrics, installation and usage instructions. The metrics that mentioned are appeared as badges, and they are special because they are not static. Instead, they are always updated by the reference to which are configured to be coupled with. CodeTour's badges that are available on README are the following:

- Build Workflow status -- *from Github Actions*
- Plugin latest version (released) -- *from JetBrains Marketplace*
- Plugin downloads/installations -- *from JetBrains Marketplace*
- Plugin rating -- *from JetBrains Marketplace*

Figure 13. CodeTour README Badges



Another important README's part is the **Plugin Description** for which there is some magic behind. Everything that is between tags `<!--Plugin Description -->` and `<!--Plugin Description end -->` is automatically retrieved and published to Marketplace as the Description of any new release. This way, there is a single point of reference for updating Plugin's description! Further information can be found on section 5.4.2.1.

5.4.2.1 Continuous Integration

Much effort has been put into Automating non-coding related things and thanks to IntelliJ Plugin Template many of them have been implemented and an overview for each one of them is provided below.

More specific, **Build** workflow is declared on build.yml file and it does the following build related tasks:

- Triggered when a new Pull Request opens or closes (merged)
- Runs the **test** Gradle task (also runs backwards compatibility with older IDE versions)
- Runs code analysis related tasks (Qodana and CodeQL)
- Runs **buildPlugin** Gradle task which also generates the executable
- Runs the **verifyPlugin** Gradle task, which validates the executable (making sure it is a valid plugin) through the IntelliJ Plugin Verifier tool
- [only on PR close] Collects the release notes from CHANGELOG.md file (everything under the **## [Unreleased]** section)

- [only on PR close] Creates a draft release on the Github release page, with the collected release notes and the attached executable

Once the Pull Request is closed, the Draft release can be promoted to public release with the hit of a single button (from Github's draft release page). This will fire the *released* event and then the **Release** workflow, declared on release.yml file, will take action:

- Triggered on *released* event
- Signs the executable of the Plugin, with a provided certificate (required for publishing the plugin to JetBrains Marketplace)
- Publishes the plugin to JetBrains Marketplace using a configured PUBLISH_TOKEN
- Patches the CHANGELOG.md file (release notes that were under **## [Unreleased]** will go under a new section with the name of the release e.g. ## [0.0.3] and will create a Pull Request for this change)

And that's it! The new release has been published to Marketplace with almost zero extra effort.

The value of this automation is priceless, as it speeds up the development process making developers to focus only on the code! This is great for any Software, and I am proud to have it done for CodeTour because it literally saves me valuable time. On each new CodeTour release, the work that I have to do is:

- Develop features
- Keep the CHANGELOG.md updated with the release notes under **## [Unreleased]** section
- Keep the Plugin Description section in README.md updated (if it needs to change)
- Bump the version (if not already bumped)
- Open a PR when feature(s) are done
- Wait for all the checks to pass (build, test, compatibility, verifier, analysis)
- Merge the PR
- Make the draft release Public
- Merge the auto created PR with the patched CHANGELOG.md

Nothing more is required! The new version has been published to JetBrains Marketplace, a Github release has been created as well as a code tag, and the code is ready for further development!

Note: Some Automation stages requires some configuration including certificates, passwords and tokens. Such configuration cannot be publicly available and so they are provided through Github ACTION SECRETS. Once configured, those secrets can be used on project's pipelines through the interpolation logic: `${{secrets.GITHUB_TOKEN}}`. The Secrets that CodeTour uses are briefly describe on the following table:

Table 5. Action Secrets

SECRET Name	**Usage**
GITHUB_TOKEN	The authentication token, automatically assigned by GitHub when a pipeline starts
CERTIFICATE_CHAIN	Certificate chain, should contain: `-----BEGIN CERTIFICATE-----...-----END CERTIFICATE-----`
PRIVATE_KEY	Certificate private key, should contain: `-----BEGIN ENCRYPTED PRIVATE KEY-----...-----END ENCRYPTED PRIVATE KEY-----`
PRIVATE_KEY_PASSWORD	Password used for encrypting the certificate file.
PUBLISH_TOKEN	Publishing token generated in your JetBrains Marketplace profile dashboard.

5.4.2.2 Issues

The planning of CodeTour development is performed using Github Issues and Kanban Boards with some simple automation. The issues are grouped into some labeled collections that are visible as columns on the Board. The columns that are currently used, along with their usage follows:

Table 6. CodeTour Kanban Board

Column	Usage	Automation
Backlog	*Issues that need to be picked up. In other words, the TODO list*	- Newly Added Issues - Reopened Issues
Sprint	*Issues that are planned for work and will be included on the next release*	-

In progress	*Issues/PR that have been picked up, and are currently in progress*	- Newly Added Pull Requests - Reopened Pull Requests
Icebox	*Issues/PR that are either blocked, or paused for any reason*	
Done	*Issues and PR that have been completed (either merged or not) or closed*	- Closed Issues - Merged Pull Requests - Closed Pull Requests (with unmerged commits)

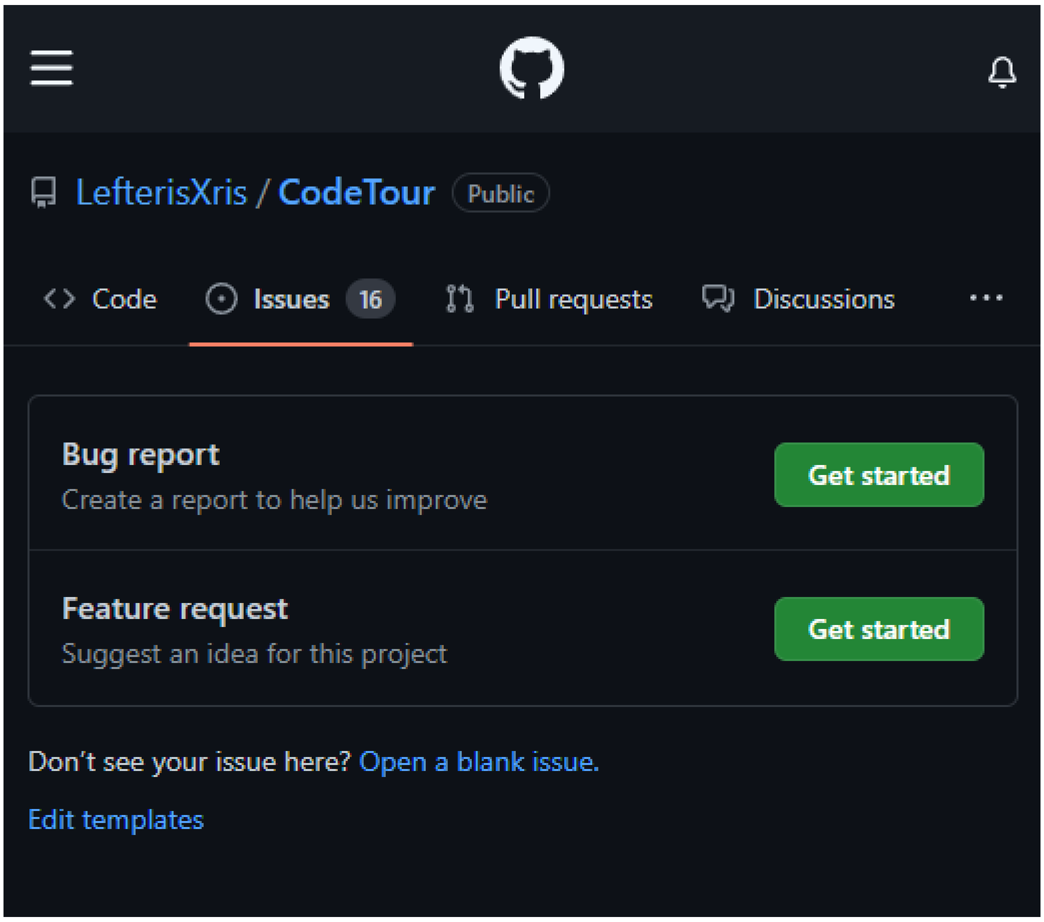
As CodeTour is Open Source, issues may be opened by anyone from the community. This sounds great, but imagine the mess that could happen if anyone opens bugs on a different way, not mentioning important info like Steps To Reproduce, or IntelliJ version etc. To avoid such cases, some Github templates have been created and can be used for opening issues, based on their type. This way there will be a consistency on project's issues which will make the maintenance more manageable.

The available templates are briefly described below:

- Bug report template (bug_report.md) should include information related to the following:
    - A clear and concise description of what the bug is
    - Steps to reproduce the behavior
    - Expected Behavior vs Actual
    - Screenshots (if applicable)
    - System (OS, IDE, version etc)
- Feature request template (feature_request.md) should include:
    - The idea of the feature through a concise description
    - The solution that you might have thought (how would you image to be or work)
    - Alternatives or other systems that support similar features

These templates are automatically show to users whenever they try to open a new Issue as show below.

Figure 14. Github Issue Templates


LefterisXris / CodeTour Public

Code Issues 16 Pull requests Discussions

Bug report
Create a report to help us improve
Get started

Feature request
Suggest an idea for this project
Get started

Don't see your issue here? Open a blank issue.

Edit templates

5.4.2.3 Discussions

A Discussion Area has been created for CodeTour for easier interaction with the community. The Discussion Area can be used to make announcements, create Polls, ask questions, share feedback, thoughts, ideas etc.

On a typical Open Source project, a Discussion Area can also be used as a Frequently Asked Questions (FAQ) section, or even forum related to any application topic. The idea of having a place where voting questions and answers is provided by default, works great for Open Source projects and I hope that this will also happen to CodeTour.

5.4.2.4 Wiki Pages

Any Open Source project, needs to have some kind of documentation. Github provides the Wiki Pages which can be used for project's documentation as it provides some nice features:

- Version control of the documentation
- Rich text support by enhanced MARKDOWN

- Online editor with Preview feature
- Easy images or attachments upload with a simple Drag & Drop
- Multiple pages and sub-pages structure with automated Table of Content for quick navigation
- Publicly available (no need to care for deploying the documentation) (multiple Easy editing

CodeTour's Wiki Pages contain a lot of information, including CodeTour's demo, tutorials, Best Practices, Technical Implementation details and some information related to Contributing on CodeTour repository.

5.4.2.5 Contributing

As CodeTour is open source, anyone is free to use, study, modify and distribute the project for any purpose. Open source let people contribute on projects they rely to and most of the time contributing is a rewarding way to learn new skills and improve collaborations skills which is probably the most important one.

To do so, CodeTour's repository is configured to include much information to make it easier for anyone interested to contribute. For this purpose, the following have been configured/created:

- Project's quick description (with topics/keywords)
- README.md -- ***tech stack***, ***detailed description***, ***installation steps***, *and* ***usage guides***
- CODE_OF_CONDUCT.md -- *for defining community policies*
- CONTRIBUTING.md -- *for listing Contributing instructions*
- License -- *MIT*
- Issue Templates -- *for consistency and maintenance purposes*
- Pull Request Templates -- *for consistency and maintenance purposes*

Building and maintaining a community is challenging, as on the one hand, the project should be Open Source friendly and on the other hand the contributors should follow community guidelines and in case they don't there should be tools and policies to detect such cases and take actions appropriately.

6 Demonstration

6.1 Installation

To install CodeTour, 3 options are available:

1. Use the Marketplace from within the IDE: **Settings/Preferences** > **Plugins** > **Marketplace** > Search for "**CodeTour**" > **Install Plugin**
2. Got to JetBrains online Marketplace [CodeTour - IntelliJ IDEs Plugin | Marketplace (jetbrains.com)]() and select **Get** or **Install to IDE**
3. Manually installation: Download the [latest release]() (zip file) and install it manually by locating the zip, and select it by following the options: **Settings/Preferences** > **Plugins** > ⚙ > **Install plugin from disk...**

6.2 Create Tours and Steps

Creating a Tour is very simple and can be done from CodeTour Tool Window. If the Tool window is not visible, open it from **View** > **Tool Windows** > **Tours Navigation**.

To create a new Tour, use the context menu (right click) of the Code Tours node in Tours tree:

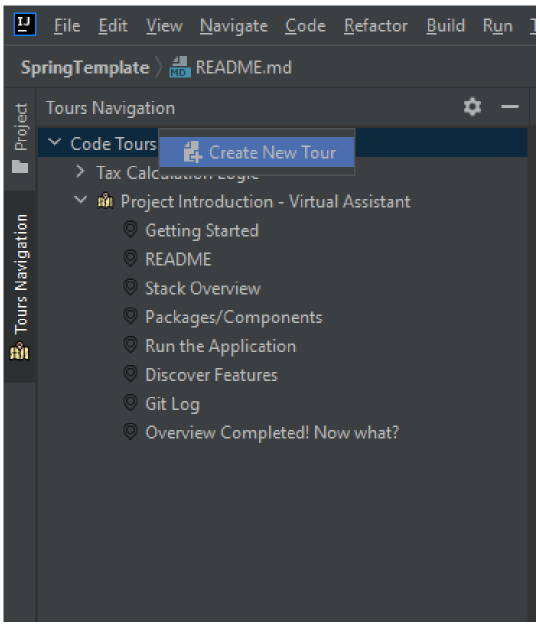


Figure 15. Create New Tour

Step creation can be done either from Tour's context menu, or directly from the Editor's Gutter context menu (similar to adding breakpoints) **Right Click** on gutter > **Add Tour Step**.

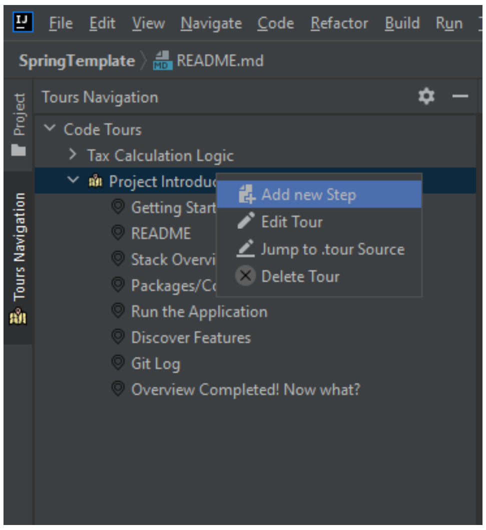
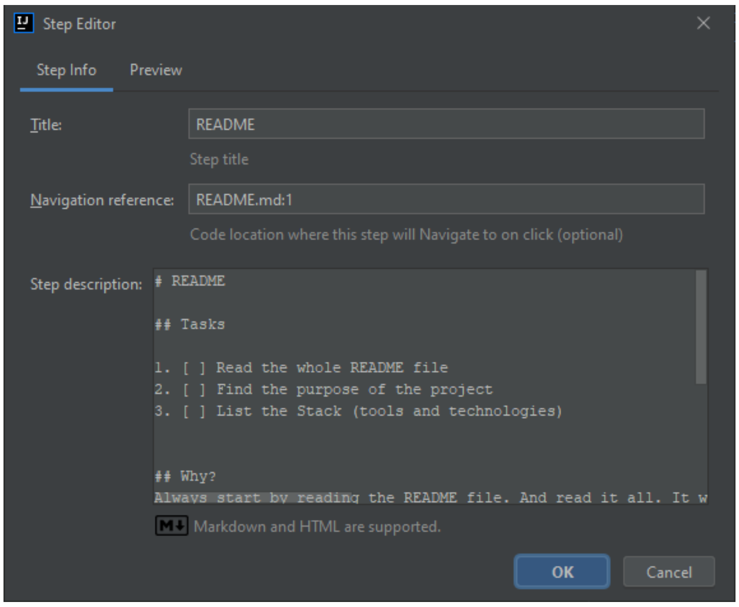


Figure 16. Create New Step

6.3 Step Editor

For creating or editing a Step, a component named **StepEditor** has been created that provides the UI to the user for managing the Step. StepEditor comes with a built-in preview mode, that renders the Step's description so that the developer can check how will it look like, once rendered.

StepEditor provides fields for Title, Navigation reference and Step's description, with full Markdown and HTML support. The editor can be seen in the following figure:



Title: README

Step title

Navigation reference: README.md:1

Code location where this step will Navigate to on click (optional)

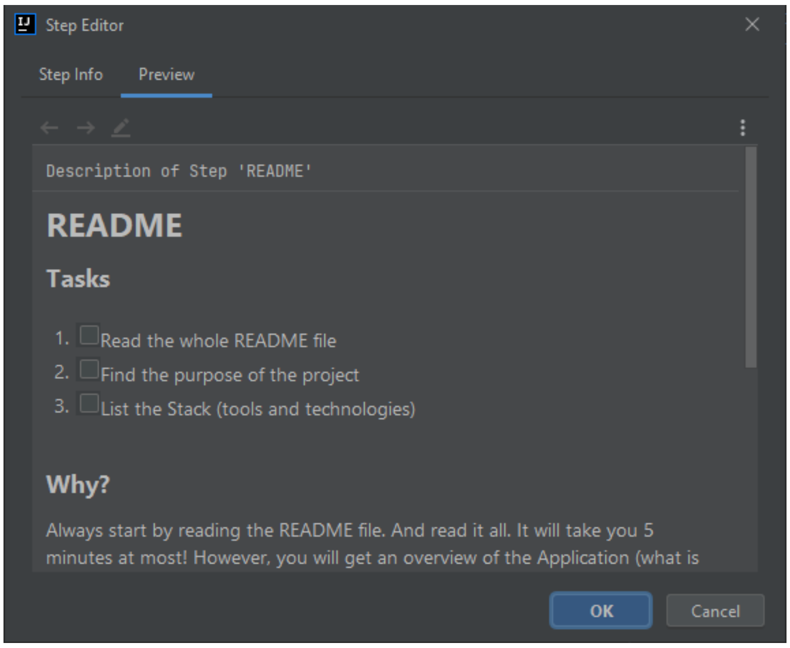
Step description:

```
# README

## Tasks

1. [ ] Read the whole README file
2. [ ] Find the purpose of the project
3. [ ] List the Stack (tools and technologies)


## Why?
Always start by reading the README file. And read it all. It w
```

Markdown and HTML are supported.

Figure 17. Step Editor UI

The rendered description of this Step is available both by preview mode and from Step Navigation and it looks like the following figure:



Description of Step 'README'

README

Tasks

1. Read the whole README file
2. Find the purpose of the project
3. List the Stack (tools and technologies)

Why?

Always start by reading the README file. And read it all. It will take you 5 minutes at most! However, you will get an overview of the Application (what is

Figure 18. Rendered Step Preview

6.4 Other Options

Steps, as nodes of Tours tree, support context menu where the options of **Edit Step**, Re-order (**Move up** and **Move down**) and **Delete Step** are available.

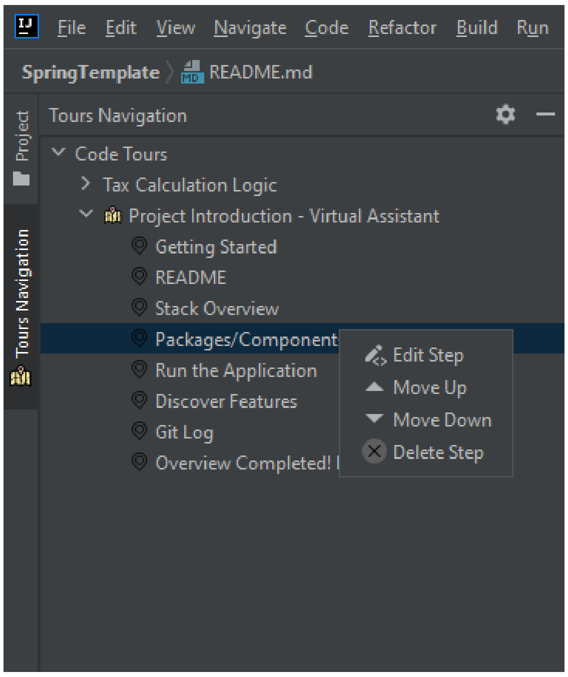


Figure 19. Step Context Menu

6.5 Use Case

To demonstrate an actual usage of CodeTour, a new sample project was implemented. The source code can be found in my Github profile.

6.5.1 Project Description

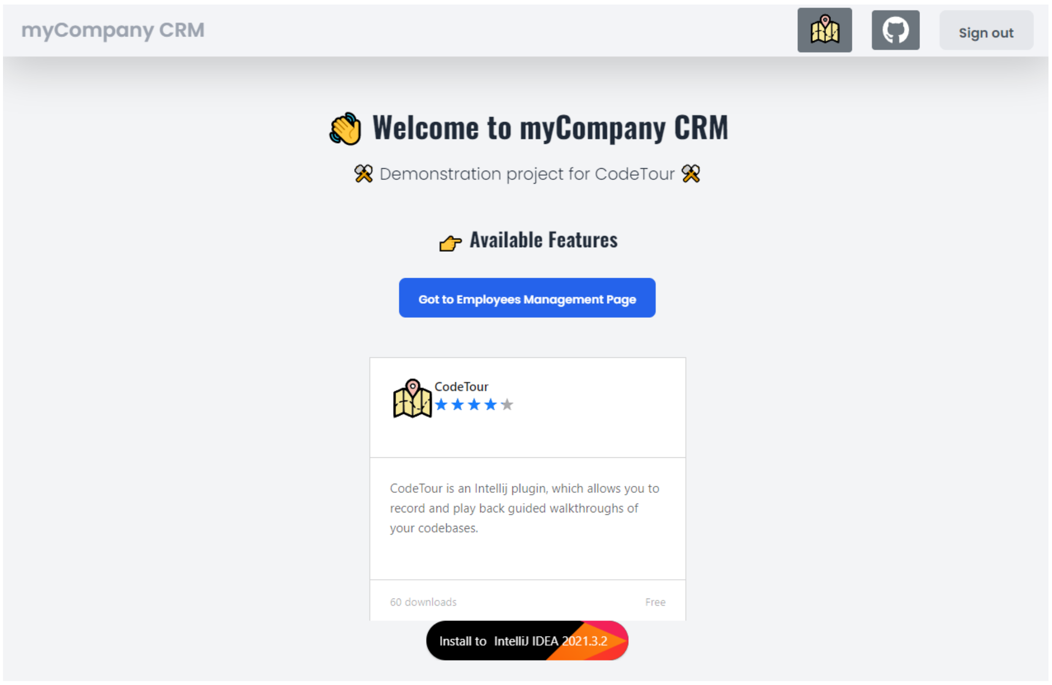
The project that was developed for demonstration purposes, is a simple Web Application, which like a small CRM (Customer Relationship Management) system provides CRUD (Create Read Update Delete) operations for **Employees** and for **Tax Regulations**. Additionally, it provides a simple **business logic**, as Tax Regulations are configurable and they define the way that tax amount is going to be calculated.

6.5.2 Stack

The project is complete web application, and consists of:

- Backend implemented with Spring Boot
- JPA for handling persistence, in H2 database (embedded)
- CRUD implementation for Employees and TaxRegulations
- Swagger Rest-API documentation using OpenAPI docs
- Business logic for Tax calculation, based on the TaxRegulation
- Tests (unit) for Tax calculation
- Frontend written using Angular
- UI with cards and forms for CRUD operations
- Trigger Tax Calculation and view calculation results
- CodeTour Promotion with download or direct installation links and Github repository reference

The main page of the Web applications is the following:



Welcome to myCompany CRM

Demonstration project for CodeTour

Available Features

Got to Employees Management Page

CodeTour

CodeTour is an Intellij plugin, which allows you to record and play back guided walkthroughs of your codebases.

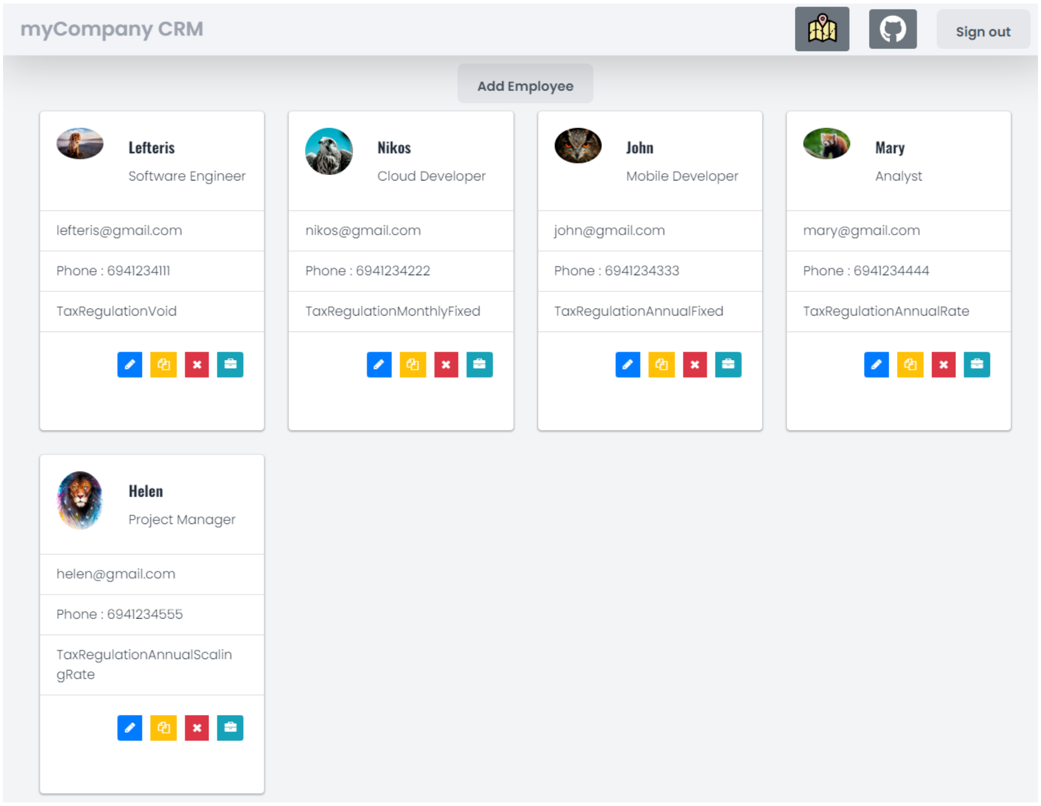
60 downloads

Free

Install to IntelliJ IDEA 2021.3.2

Figure 20. Demo Main Page

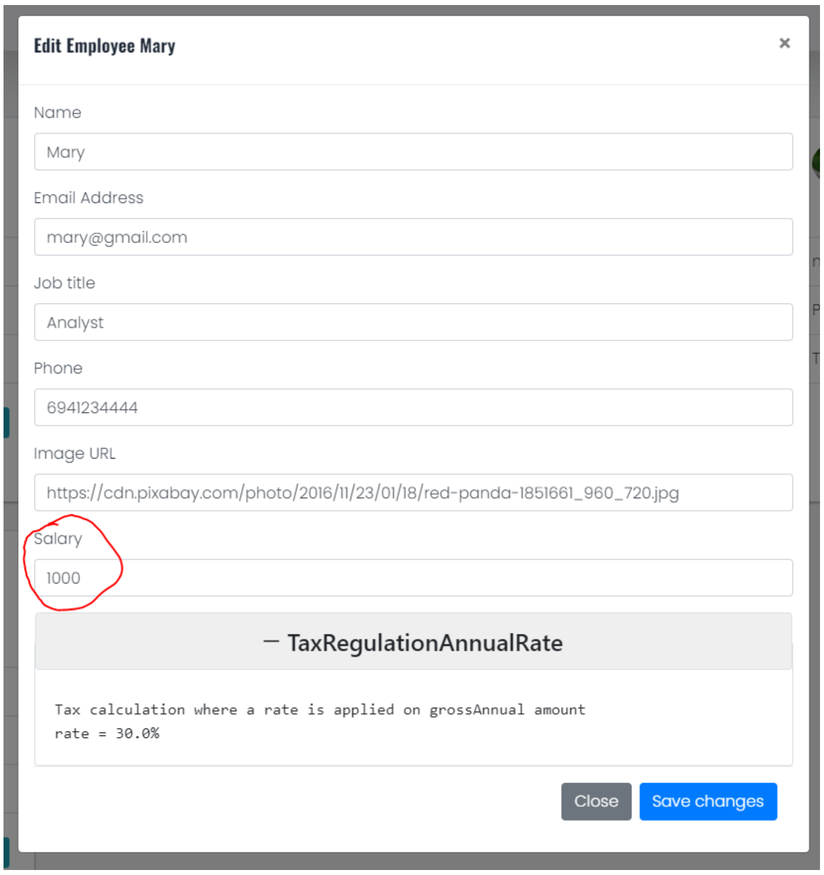
The Employees page, lists the available employees using a card-styled form and providing shortcut actions (**Edit**, **Clone**, **Delete**, **Calculate Tax**) for each employee:



Name	Role	Email	Phone	Tax Regulation
Lefteris	Software Engineer	lefteris@gmail.com	Phone : 6941234111	TaxRegulationVoid
Nikos	Cloud Developer	nikos@gmail.com	Phone : 6941234222	TaxRegulationMonthlyFixed
John	Mobile Developer	john@gmail.com	Phone : 6941234333	TaxRegulationAnnualFixed
Mary	Analyst	mary@gmail.com	Phone : 6941234444	TaxRegulationAnnualRate
Helen	Project Manager	helen@gmail.com	Phone : 6941234555	TaxRegulationAnnualScalingRate

Figure 21. Demo Employees Page

CRUD operations are provided with forms on modal dialogs. Besides the basic information of an employee, the interesting fields are the **Salary** as the tax will take it into account during calculation, and the **TaxRegulation Type** which is expandable to provide the description of its logic.


Edit Employee Mary

Name: Mary

Email Address: mary@gmail.com

Job title: Analyst

Phone: 6941234444

Image URL: https://cdn.pixabay.com/photo/2016/11/23/01/18/red-panda-1851661_960_720.jpg

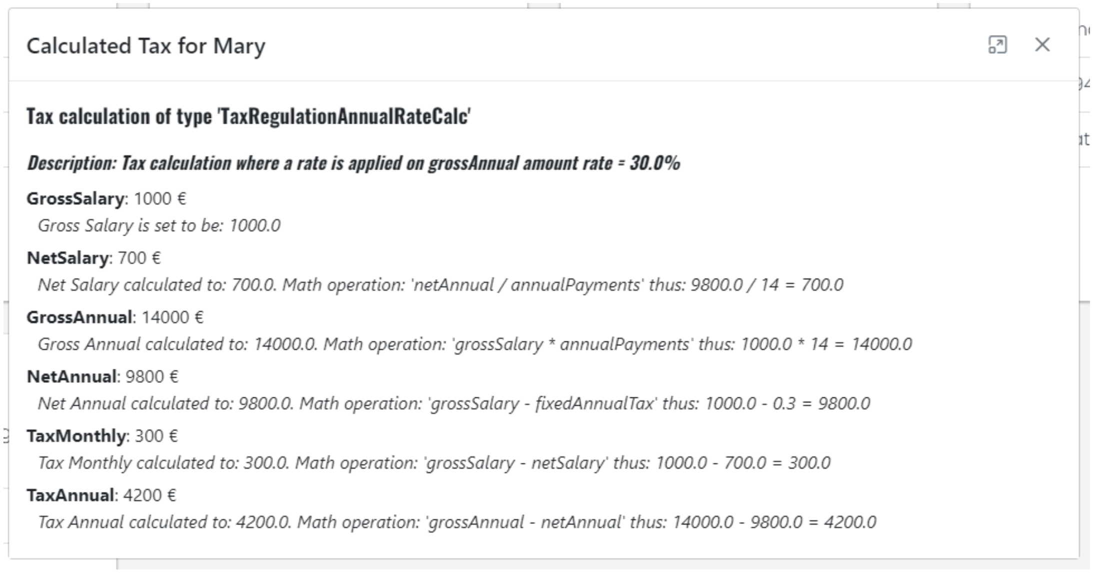
Salary: 1000

− TaxRegulationAnnualRate

```
Tax calculation where a rate is applied on grossAnnual amount
rate = 30.0%
```

Figure 22. Demo Employee Edit Dialog

The triggering of business logic (the Tax Calculation) is performed through the light green/blue button with the briefcase. The results of a calculation are shown in a modal dialog, in a simple text form:


Calculated Tax for Mary

Tax calculation of type 'TaxRegulationAnnualRateCalc'

Description: Tax calculation where a rate is applied on grossAnnual amount rate = 30.0%

GrossSalary: 1000 €
Gross Salary is set to be: 1000.0

NetSalary: 700 €
Net Salary calculated to: 700.0. Math operation: 'netAnnual / annualPayments' thus: 9800.0 / 14 = 700.0

GrossAnnual: 14000 €
*Gross Annual calculated to: 14000.0. Math operation: 'grossSalary * annualPayments' thus: 1000.0 * 14 = 14000.0*

NetAnnual: 9800 €
Net Annual calculated to: 9800.0. Math operation: 'grossSalary - fixedAnnualTax' thus: 1000.0 - 0.3 = 9800.0

TaxMonthly: 300 €
Tax Monthly calculated to: 300.0. Math operation: 'grossSalary - netSalary' thus: 1000.0 - 700.0 = 300.0

TaxAnnual: 4200 €
Tax Annual calculated to: 4200.0. Math operation: 'grossAnnual - netAnnual' thus: 14000.0 - 9800.0 = 4200.0

Figure 23. Demo Tax Calculation Results Dialog

6.5.3 Business Logic

To make the demo project more realistic, some Tax Regulation types have been implemented and are listed below:

Table 7. Tax Regulation Types

Type	**Variables**	**Description**
AnnualRate	(double) rate	Tax calculation where a rate is applied on grossAnnual amount
	Tax = Annual Gross Salary * [rate]	
AnnualFixed	(double) fixedAnnualTax	Tax calculation where a fixed amount is applied on grossAnnual amount
	Annual Net Salary = Annual Gross Salary **-** [fixedAnnualTax]	
MonthlyFixed	(double) fixedMonthlyTax	Tax calculation where a fixed amount is applied on grossAnnual amount
	Net Salary = Gross Salary – [fixedMonthlyTax]	
AnnualScalingRate	set of ScalingRateEntries ScalingRateEntry: [from-to: rate]	Tax calculation where the tax is calculated based on the scaling rates
	Tax = sum of scaling taxes e.g. 0€-8.000€ → 15%, 8,001€-12,000€ → 25% and gross salary 10.000€, tax would be 8000*15%+2000*25%=1500+500=2000€	
VoidCalc	-	Void (NoOp) Tax calculation. gross=net (wish it was true)

The class diagram of the calculation implementation is provided below, in order to have a high level overview of its structure.

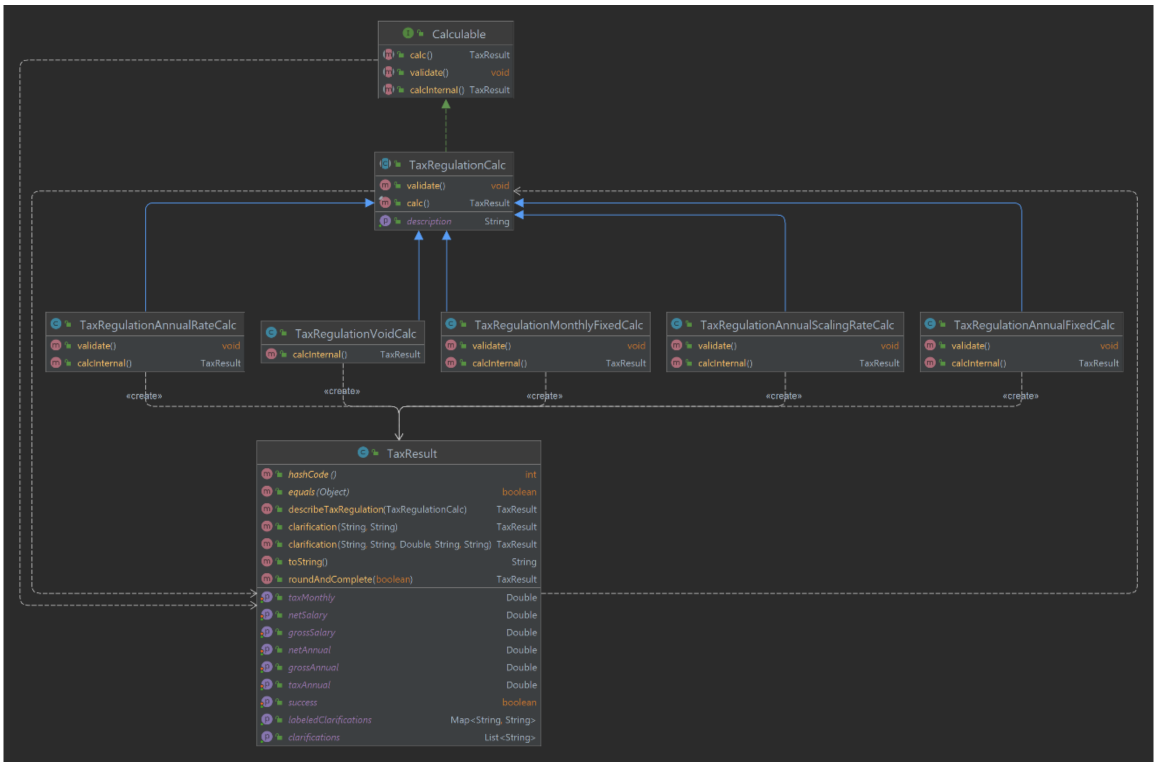


Figure 24. Demo Tax Calculation Class Diagram

The results of a tax calculation include the tax amounts for annual and monthly base as well as the net and gross salary. More specifically, the amounts are: **grossSalary**, **grossAnnual**, **netSalary**, **netAnnual**, **taxMonthly**, **taxAnnual**. Apart from these amounts, the proofs of the calculations are collected as clarifications and are available to the client results.

6.5.4 Goals

Having such a project, which contains CRUD operations and some business logic, is a good way to **test** CodeTour's efficiency on onboarding process and in feature explanation.

This project can also be used as a point of reference for **ideas** of useful Tours to be included on other projects, and **demonstration** (navigation and description structure) to showcase the supported features and any upcoming feature.

In the current state, only the demonstration will be utilized. In the future, this project can be further developed and used for performing use cases in Software

Engineering students in order to collect their feedback and check whether CodeTour helped them understand the project and compare it with others that didn't use CodeTour and tried to onboard on this project with other ways.

6.5.5 Tours

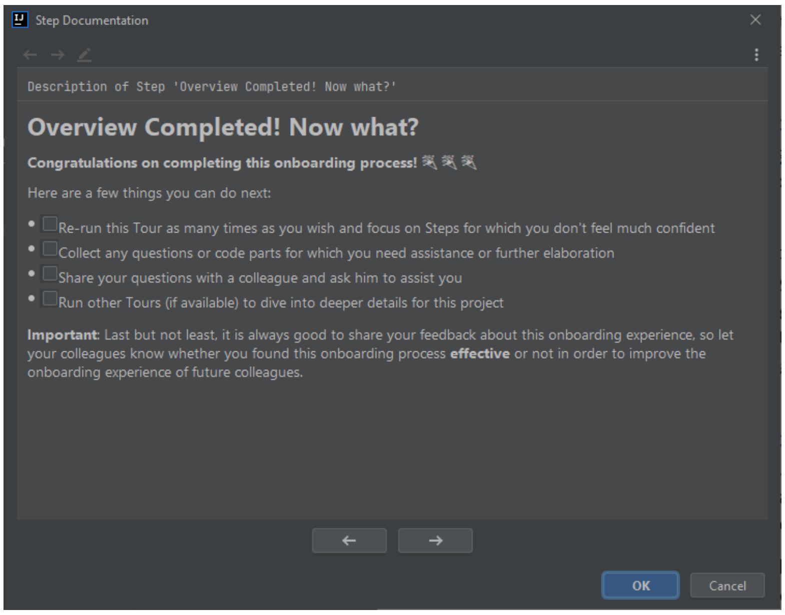
For the purpose of the demonstration, 2 Tours have been created: Project Introduction - Virtual Assistant and Tax Calculation Logic

6.5.5.1 Project Introduction - Virtual Assistant

This Tour simulates a Tour for onboarding new joiners on a project. It should be the first thing that a new joiner would do, as its goal is to guide the new joiner to follow a specific approach for onboarding, in order to gain a brief understanding of the project in a reasonable time.

The Tour consists of 8 Steps that each one has some Tasks to be done, some Takeaways as Step's sum up, and the elaboration of the purpose and the significance of this Step. A short description of each step follows:

1. **Getting Started**: General information about the Tour, its purpose and directions on how to use it.
2. **README**: Navigates to README.md file. Contains 3 tasks and 2 takeaways
3. **Stack Overview**: Prompts the reader to explore the Stack of the project. Contains 3 tasks and 3 takeaways
4. **Packages/Components**: Prompts the reader to read the files of the project to identify the structure. Contains 3 Tasks and 4 takeaways.
5. **Run the Application**: Running the application and tests
6. **Discover Features**: Playing around with the features to identify how they are implemented.
7. **Git Log**: Taking an idea of how the team works with git
8. **Overview Completed! Now what?**: Next Steps and ideas for better onboarding


Step Documentation

Description of Step 'Overview Completed! Now what?'

Overview Completed! Now what?

Congratulations on completing this onboarding process! 🎆🎆🎆

Here are a few things you can do next:

- ☐ Re-run this Tour as many times as you wish and focus on Steps for which you don't feel much confident
- ☐ Collect any questions or code parts for which you need assistance or further elaboration
- ☐ Share your questions with a colleague and ask him to assist you
- ☐ Run other Tours (if available) to dive into deeper details for this project

Important: Last but not least, it is always good to share your feedback about this onboarding experience, so let your colleagues know whether you found this onboarding process **effective** or not in order to improve the onboarding experience of future colleagues.

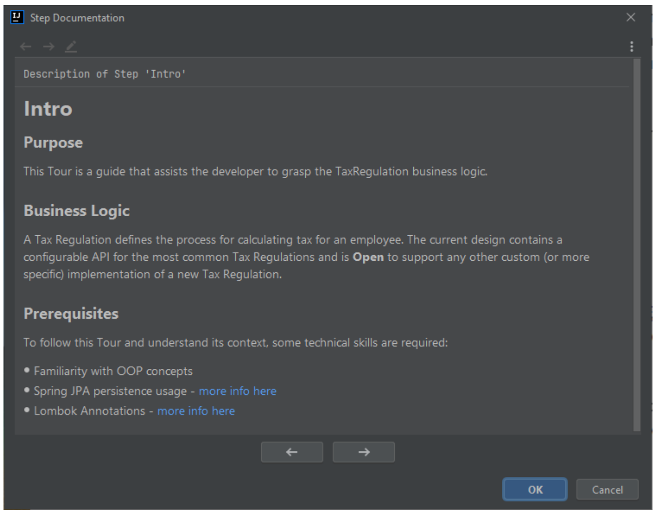
Figure 25. Demo Tour 1 Step

6.5.5.2 Tax Calculation Logic

This Tour simulates an explanation of a feature that could have been done in the context of a Pull Request.

The Tour consists of 6 Steps, and they try to assist the reader understand the tax calculation logic easier. The steps along with a short description follows:

1. **Intro**: states the Purpose of the Tour, the business logic that was implemented and the prerequisites for the implementation:


Step Documentation

Description of Step 'Intro'

Intro

Purpose

This Tour is a guide that assists the developer to grasp the TaxRegulation business logic.

Business Logic

A Tax Regulation defines the process for calculating tax for an employee. The current design contains a configurable API for the most common Tax Regulations and is **Open** to support any other custom (or more specific) implementation of a new Tax Regulation.

Prerequisites

To follow this Tour and understand its context, some technical skills are required:

- Familiarity with OOP concepts
- Spring JPA persistence usage - more info here
- Lombok Annotations - more info here

Figure 26. Demo Tour 2 Step

2. **Tax Regulation Entity**: provides info about the abstraction of the Tax definition (**TaxRegulation**) as well as for the calculation (**TaxRegulationCalc**), and it ends

with 2 Tasks at the end which prompts the reader to navigate to sub-classes to see the implementations.

3. **Calculation Logic**: defines the Calculable interface where the main methods are declared. States the reasons and what should be implemented by sub-classes. Ends with 3 Tasks, 2 of which are related to calculation implementation and the last one is a prompt to check the relationship of TaxRegulation with the Employee entity (how they are linked).
4. **Calculation Result**: briefly describes the results class. Prompts 3 Tasks to check TaxResult fields and to explore the clarifications API.
5. **Trigger Calculation**: contains information about the entry point of the calculation, i.e. how can someone trigger the calculation and what is that execution path.
6. **Calculation Tests**: prompts the reader to check the unit tests related to Tax Calculation logic

7 Conclusion

The idea of a Tool that is able to explain the code to a developer with interaction and hands on code navigation, without the need to rely on others seems great! Such a tool can be very useful for some cases, but there are also some big challenges.

7.1 Ideal Usages

The first case where CodeTour could be very useful is the **Onboarding** process. Just consider that a more Senior developer prepares some Tours in which he explains the basic components of a project, a more detailed view of important code parts and also some common pitfalls. Those Tours could be then shared with the new joiners, so that they would have the ability to interactively watch these tours and navigate on the code, having the Senior's explanation on their side to guide them throughout the code. Those Tours can be viewed as many times as needed by one to grasp them, and navigation on Steps can be played or rewind based on individual's pace.

Tours can include anything. However, the following Tours are considered very useful for onboarding processes:

1. **Project Intro - Virtual Assistant**
   A tour that will guide the new joiners on what to look and where to find them in the project, with steps in and order that will optimize the onboarding time
2. **Project Conventions**
   Conventions of a project can be mentioned through a Tour. This tour would include the statements or the links to the conventions, and can navigate to real code examples.
3. **Framework Basics and Resources**
   If the project uses a specific framework (like Spring Boot, Angular Quarkus etc) it's a good idea to have a Tour that will provide information about that framework to those who are completely new to it. Apart from the information, the Tour could provide real examples from the code, and at the end, any extra resources related to that framework can also be shared.

Another case where CodeTour could be very useful is in the context of a **Pull Request** (PR). When a new feature is implemented and a PR is opened, in a typical

Development workflow a code Review is taking place next. Understanding a code either it is new or it is just modified, is sometimes a challenging task as the reviewer has to think the same way the developer did when implementing the feature and evaluate whether it is:

- conventional (if follows specific rules, conventions and style of the team, if reuses same structures/components etc)
- functional (if the code changes are actually dealing with the PR issue, either to resolve the related bug or implement the related feature
- efficient (if the logic is performant, or there was another easier way)

As you may understand, code review is not easy and requires time. Consider that you are assigned to review a PR related to a new feature implemented by a colleague that you don't know you can't directly talk to them. How much time do you think you would need, to make a good Code Review of that PR? Wouldn't be easier, if the developer who implemented the feature, attached a Tour (or a couple of them) in which they described their logic and provided a guided navigation on the changes they performed so that you could instantly checkout the code and start watching developer's code changes along with the provided explanation/description? For such cases, CodeTour can be very valuable!

Last but not least, as mentioned before Tours can be version controlled and thus, they could be maintained throughout the development process. This means that every time the code changes, any related Tour should also be updated to match the new code and be always valid. That way, a project can use Tours as a way to Document the Features it supports so that others can always view the Tours that are related with some features in order to understand how that feature is implemented on the code. So, CodeTour can be a good solution for **Feature Documentation**, if Tours are version controlled and their maintenance is included on Development process in order to be always in-sync with the code.

7.2 Challenges

Although CodeTour offers some great features, there are some challenges that could potential make someone refrain from using CodeTour.

The challenges are mostly related to **maintenance**, as if a team chooses to use CodeTour as their Feature Documentation tool, although the added value on project's comprehensibility would be high, there will be some cost, in terms of time required to update and maintain the Tours in order to be always in-sync with the code base. This is a very big challenge because in a real-world projects there is rarely enough time for such maintenance related tasks and as a result, Tours will soon become outdated and probably not useful anymore. This happens very frequently as higher-level Managers have not clear view of Development process and they always expect new features or changes leaving no-time for such tasks, something that eventually leads to other important issues like Technical Dept.

Another challenge related to maintenance is **Step reference and dynamic updates**. Each Step in CodeTour is associated with a file:line which indicates the location where it will navigate the developer. In future versions, CodeTour will support selecting code blocks instead of just file:line (already supported on VS Code CodeTour). However, the challenge still remains the same in both cases: What if a developer add/remove lines from a file which is included on a Step? What if a developer remove/update the code block that was referenced by a Step? Could the Steps be auto adjusted to match the updated code? Dynamic update seems very difficult in both cases, as in the first scenario (file:line) Steps should be adjusted on every line change (with some proper logic of course) but without having being coupled to code context how should they adjust? For instance, a method that was initially a Step's reference, may be moved at the end of the file making the Step invalid (wrong line) and as it is not aware of the method, how should it be automatically updated? On the other scenario (code block), code blocks are very likely to change and if they do, should Step be updated to include the whole new block? What if some more code will be placed in between? That kind of changes can be caught and maybe some warnings or popups could be thrown to the developer to warn him about the state of the Step, but this doesn't seem so nice and it will probably be very annoying. Ultimately, maybe a combination of those two scenarios could work, but further investigation is required and probably a more complex implementation will take place.

Those maintenance related challenges are actually the reasons due to which CodeTour may be preferable to be used ad-hoc and once in a while, instead of being part of the project. Ad-hoc or once in a while cases may be Project Onboarding and PR review.

Finally, **Compatibility** between CodeTour and VS Code CodeTour may also considered a challenge as it would be very nice for Tours to work on both IDEs regardless the IDE they generated from. The challenge relies mostly on the structure and the API that each IDE provides as in both cases the Tour schema should be the same, but be functional either in IntelliJ-based IDEs or in VS Code. In addition, some extra features are about to be implemented on CodeTour and thus keeping compatibility between both projects will require some communication and alignment and I am not even sure if it’s possible. However, it would be nice to have it.

7.3 Future Work

There are many related fields for which further research may be done, as well as more ways to evaluate CodeTour. The ideas and suggestions are listed below as potential Research Questions (or just topics).

7.3.1 Academia Research

Q1. What is the average time that a developer needs to onboard to a project (per level probably) and what could affect that time?

Q2. Experiments with Eye-tracking devices to study code navigation patterns on project onboarding. (and probably identify best patterns?)

Q3. Could the extensive usage of OOP (Object Oriented Programming) affect negatively the project’s Comprehensibility? (e.g. having multiple levels of inheritance or overdoing method overloading could make a project more difficult to read, especially to more junior developers)

7.3.2 CodeTour Evaluation

Q1. What kind of Tours would professional developers create? Are they maintained/evolved?

Q2. What’s the overhead of maintaining such tours?

Q3. Could tools like CodeTour automatically maintain Tours during code base changes?

Q4. Experiment (Case Study). Onboard similar-level developers (or students) on the same project, with and without CodeTour and evaluate the comprehensibility through Questionnaires (to check if the understood it correctly) and Code Tasks (to check if adding new features becomes easier/faster for those onboarded using CodeTour).

Q5. Experiment. Do a real Pull Request on an open source project explaining the logic of the implementation (PR context) using a Tour (or many) and asking from the reviewer to use CodeTour to review the PR.

8 References

Anda, B. *et al.* (2006) "Experiences from introducing UML-based development in a large safety-critical project," *Empirical Software Engineering*, 11(4), pp. 555–581. doi:10.1007/s10664-006-9020-6.

Cazzola, W., Ghoneim, A. and Saake, G. (2006) "Viewpoint for maintaining UML models against application changes," in *ICSOFT 2006 - 1st International Conference on Software and Data Technologies, Proceedings*. INSTICC Press, pp. 263–268. doi:10.5220/0001310802630268.

Curtis, B. *et al.* (1989) "Experimental evaluation of software documentation formats," *The Journal of Systems and Software*, 9(2), pp. 167–207. doi:10.1016/0164-1212(89)90019-8.

Fernández-Sáez, A.M., Genero, M. and Chaudron, M.R.V. (2013) "Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study," *Information and Software Technology*. Elsevier B.V., pp. 1119–1142. doi:10.1016/j.infsof.2012.12.006.

Gan, K. and Zhang, C. (2010) "Effect of employees' turnover rate on industrial firms' production efficiency: A conceptual model," in *2nd International Conference on Information Science and Engineering, ICISE2010 - Proceedings*, pp. 2813–2817. doi:10.1109/ICISE.2010.5691528.

Garousi, G. *et al.* (2013) *Evaluating Usage and Quality of Technical Software Documentation: An Empirical Study*.

Hebig, R. *et al.* (2016) "The quest for open source projects that use UML: Mining GitHub," in *Proceedings - 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2016*. Association for Computing Machinery, Inc, pp. 173–183. doi:10.1145/2976767.2976778.

Jones, S.J. and Burnett, G.E. (2007) "Spatial skills and navigation of source code," *ACM SIGCSE Bulletin*, 39(3), pp. 231–235. doi:10.1145/1269900.1268852.

Lehman, M. (1996) *Laws of Software Evolution Revisited*.

Muhammad KHAN, D. *et al.* (2015) *The Impact of Employees' Turnover at the Productivity of a Software Machines Learning Approaches for Real-world Problems View project Performance based comparison between RDBMS and OODBMS View project The Impact of Employees' Turnover at the Productivity of a Software, Article in International Journal of Natural and Engineering Sciences*. Available at: www.nobel.gen.tr.

Nassif, M. and Robillard, M.P. (2017) "Revisiting turnover-induced knowledge loss in software projects," in *Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*. Institute of Electrical and Electronics Engineers Inc., pp. 261–272. doi:10.1109/ICSME.2017.64.

Pinto, G. *et al.* (2019) "Training software engineers using open-source software: The students' perspective," in *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET*

2019. Institute of Electrical and Electronics Engineers Inc., pp. 147–157. doi:10.1109/ICSE-SEET.2019.00024.

Plosch, R., Dautovic, A. and Saft, M. (2014) “The value of software documentation quality,” in *Proceedings - International Conference on Quality Software*. IEEE Computer Society, pp. 333–342. doi:10.1109/QSIC.2014.22.

Schulz, B. (2008) *The Importance of Soft Skills: Education beyond academic knowledge, NAWA Journal of Language and Communication*.

Smith, T. *et al.* (2014) “Selecting Open Source Software projects to teach software engineering,” in *SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, pp. 397–402. doi:10.1145/2538862.2538932.

Spinellis, D. (2018) “Being a Software Developer,” *IEEE Software*. IEEE Computer Society, pp. 4–7. doi:10.1109/MS.2018.2801555.

Storey, M.A. *et al.* (2007) “How programmers can turn comments into waypoints for code navigation,” in *IEEE International Conference on Software Maintenance, ICSM*, pp. 265–274. doi:10.1109/ICSM.2007.4362639.

Valentino Vranić *et al.* (2015) *Challenges in preserving intent comprehensibility in software*. Available at: https://www.researchgate.net/publication/286875300_Challenges_in_preserving_intent_comprehensibility_in_software.

9 Appendix – Useful Resources

- CodeTour JetBrains Marketplace: CodeTour - IntelliJ IDEs Plugin | Marketplace
- CodeTour Github repository: LefterisXris/CodeTour: IntelliJ Plugin - MSc Thesis
- Survey Platform: SurveySparrow Surveys
- Survey domain: https://codetour.surveysparrow.com/
- Survey Questions: Survey - Code Reading Challenges & Best Practices.pdf
- Survey Results: Results Report - Code Reading Challenges & Best Practices.pdf
- DeveloperWeek Conference Session: Eleftherios Chrysochoidis from Accenture at DeveloperWeek 2022