universität wien

BACHELORARBEIT / BACHELOR’S THESIS

Titel der Bachelorarbeit / Title of the Bachelor’s Thesis

„Eye-Gaze to Source Mapping and Navigation Pattern Extraction - A Proof-of-Concept“

verfasst von / submitted by

Benjamin Hartmann BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Bachelor of Science

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 033 526
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Bachelor Business Informatics
Betreut von / Supervisor:	Univ.-Prof. Dr. Uwe Zdun
Mitbetreut von / Co-Supervisor:	Dr. Georg Simhandl

Contents

Abstract

Many organizations are implementing their applications as Micro-Services nowadays, thus moving away from monolithic architectures. But how are developers coping whith learning new architectural styles? Controlled eye-tracking experiments where the participants had to solve various programming tasks within Micro-Services and Monolithic architectures are the base of this thesis which tries to provide a tool for automated extraction of the used source code, mapping it to the input data and visualize it for further analysis. This tool should act as a proof-of-concept-solution, enabling the path for further work to gain a better understanding of the difficulties that can arise with architectural styles that are not well known to the developer. The results of this thesis proof, that using eye tracking in combination with automated tools for analyzing and visualization of the data, deeper insights can be gained in how developers of different experience levels are navigating through and reading source code. Furthermore, differences and patterns used when working within different architectural styles could be part of future work, making usage of the provided tool.

1 Introduction

Developing software has always been non-trivial, but the lightning speed in which new technologies evolve within the field of software development makes it even more difficult for professionals to keep up with their knowledge. The need for continuous learning within the context of developing and implementing software stands out independently from the used programming language. A quick look at the statistics of maven's central repository [1], reveals that there are 6,369,250 artifacts accessible for use when developing software within the Java ecosystem at the moment of writing this thesis [7]. Clearly, this makes a good example of how many frameworks, libraries and other tools are available to work with and this number does not took much consideration of languages outside of the Java ecosystem, like C++, JavaScript or other languages that are not using the Java Virtual Machine (JVM). Because there are so many frameworks in use, it is to argue that a good software developer should have profound knowledge of at least some of them and should also be prepared to gain knowledge about future developments within this area. But there is another aspect of software developing besides the comprehension of various languages or frameworks and that is Software Architecture, which also requires continuous learning and understanding from the developer. Chapter 3 of this thesis is focused on introducing Service-Oriented Architectures and Micro-Services. Followed by that, the practical part of this thesis will be discussed. The last chapter summarizes the conclusions found and gives an outlook for possible future work. To understand the concepts and techniques used by the software architectures within

[1] Maven is a build automation tool used for managing Java-based projects and Maven's Central Repository is widely used for dependency management and publishing artifacts to be used by others.

this thesis, a quick introduction to the history of software architecture is given within the next section.

2 History of Software Architecture

The first occurrence of the term 'software architecture' is documented in a written report about the 1969 NATO Conference on Software Engineering Techniques [12]. It was Ian P. Sharp who introduced it with the following words:

> I think that we have something in addition to software engineering: something that we have talked about in small ways but which should be brought out into the open and have attention focused on it. This is the subject of software architecture. Architecture is different from engineering. I believe that a lot of what we construe as being theory and practice is in fact architecture and engineering; you can have theoretical or practical architects: you can have theoretical or practical engineers. I don't believe for instance that the majority of what Dijkstra does is theory—I believe that in time we will probably refer to the "Dijkstra School of Architecture" [12].

Software architecture as a distinct discipline did not found its ways into further discussions before 1990, although Ian P Sharp's remarkable statements. Prior that, 'architecture' was mostly used for describing the hardware structure of a computer system or on which instruction set families a computer can operate on, the evolving of software architecture started around 1992 and is continuously bringing out interesting ideas (cf. Figure 1)[18]. The seminal paper 'Foundations for the Study of Software Architecture', published 1992 by Dewayne E. Perry and Alexander L. Wolf , can be seen as an important milestone in the rising of software architecture as an independent part within software development besides the field of software engineering [24]. The authors identified three core parts which form the model of software architecture: *Elements, Form, Rationale.*

- The *elements* of a software architecture can be distinguished in

    1. *Processing elements* that are used to apply transformation on *processing elements.*
    2. *Data elements* that are containing the data to be transformed by the *processing elements*
    3. *Connecting elements* that are acting as a glue to hold other architectural pieces together and act at times either as *processing elements* or *data elements*

  Perry and Wolf provided a real world example taken from the world of sports to paint a more lively picture of the elements described above. Water Polo is played by players which act as the *processing elements*, the

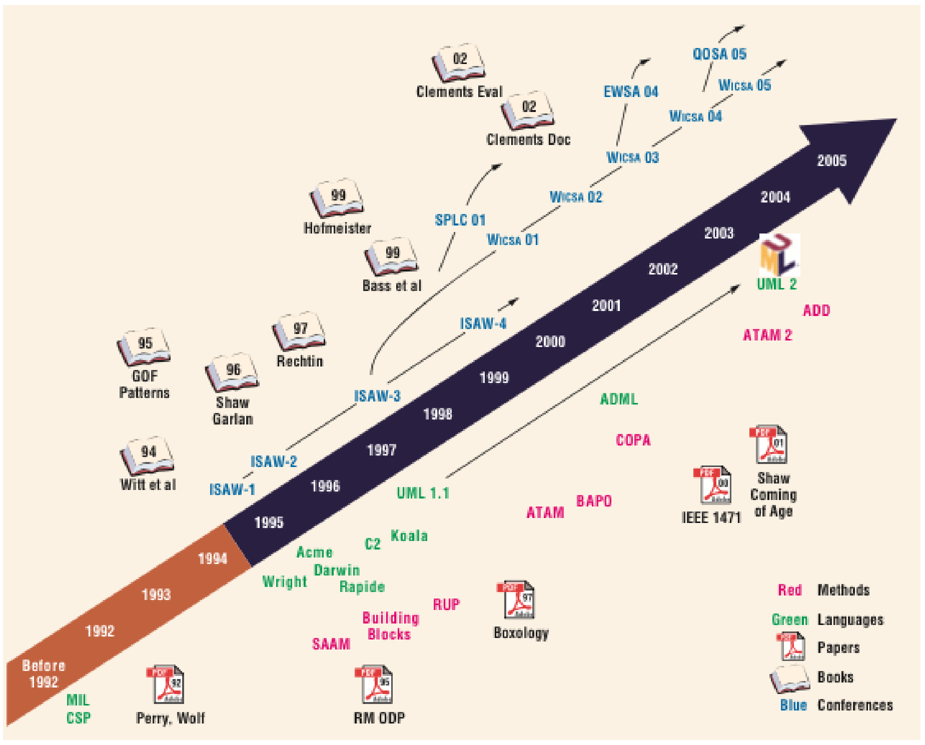


Figure 1: History of Software Architecture 1992 - 2005 (Kruchten et. al. 2006)

ball can be seen as a *data element* whereas the water takes the part of the *connecting element*. Furthermore if water polo is compared with polo and soccer, the similarities are obvious and there is only the *connecting element* to be identified as main difference and thus the connecting elements mark an important role in distinguishing between different software architectures.

- The *form* of software architecture, as described by the authors in [24], consists of weighted *properties* and *relationships*, whereas the weighting either stands for the importance of the specific *property* or *relationship* or for the necessity of selecting among alternatives.

    1. *Properties* act as constraints for choosing architectural *elements*, they define constraints on the *elements*. The *properties* define the bare minimum constraints which means that everything not mentioned within these has no restriction of *form* whatsoever.
    2. *Relationships* are constraining the placement of the used architectural *elements*, how they interact with each other as well as how the *elements* form their organizational structure. The same principal of minimum constraints used by *properties* also applies to *relationships*.

- *Rationale* is the third part of the software architecture model proposed in [24].*Rationale* acts as the motivation for choosing the *elements*, *form* and *style* of software architecture. It paints an overall picture of all these elements and explains the satisfaction of the system constraints like functional constraints, or economics, performance and reliability, to mention some of the non-functional aspects a system implementation must take into consideration.

Beneath this proposal for a formal definition of a model for creating software architectures, Perry and Wolf identified some important benefits derived from software architecture. Software architecture provides a framework, both for project managers and developers, to satisfy requirements and to design and implement the system. Even more important from the developers view is the possibility to reuse components from existing systems. Reusing components at the architectural level is freeing the developer from reinventing and building architecture components every time, thus letting him focus on more important aspects of the implementation [24]. Existing architectural components with minimum constraints and standardized structure can be reused in further projects and can truly safe a lot of effort and speed up the process as the authors conclude in their own words:

> [...] perhaps the reason for such slow progress in the development and evolution of software systems is that we have trained carpenters and contractors, but no architects. [24]

The formula *Elements, Form, Rationale* provided the basis for further scholarly debate and discussions within important works on software architecture. A notable example is the *4+1 View Model of Software Architecture*, proposed by Philippe Kruchten [19]. The model has been widely recognized not only within academic discussions, but was also used by rational consultants within large industrial projects [18]. The views introduced by Kruchten within his Architectural Blueprint are distinguished in *Logical Architecture*, describing functional requirements of the system, *Process Architecture*, describing non-functional requirements like performance, availability and concurrency issues, *Development Architecture*, which is focused on the organizational structure of the software like modules, packages, subsystems, libraries etc., *Physical Architecture*, describing non-functional requirements similar to the process architecture but focused on physical infrastructure or so called nodes, and as a fifth view, there exist *Scenarios*, putting together the other views and serving as an abstraction for the most important requirements and used for discovering architectural elements and to illustrate and validate the architectural design decisions (cf. Figure 2)[19]. Within each view, Kruchten is applying the formula *Elements, Form, Rationale* introduced by Perry and Wolf [24], describing the elements, form and patterns, the rationale and the constraints of each view independently. Splitting the description of an architecture into separate views enables to address the concerns of the different stakeholders of a project (developers, customers, data specialists...) and allows to specify different architectural styles

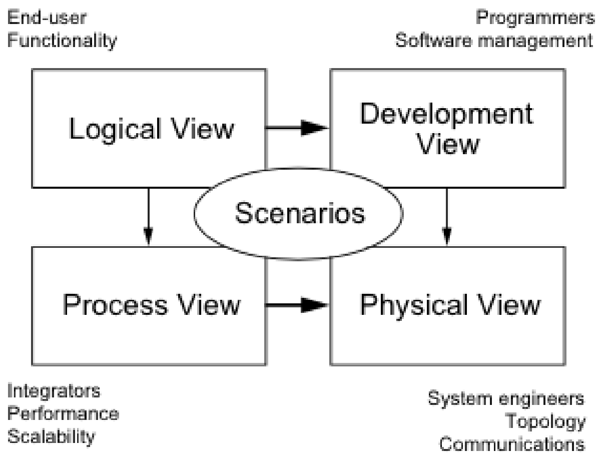
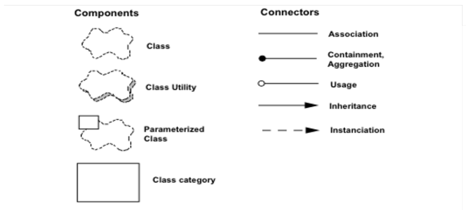
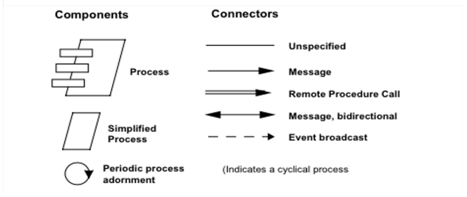


Figure 2: 4+1 View model (Kruchten 1995)

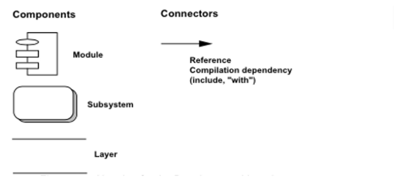
for each view. The model can be used in a generic way as Kruchten is not constraining the views to any notation or specific tools but only recommending using the same notation and tools for creating the views (cf. Figure 3). The



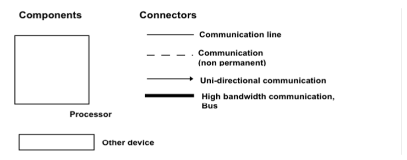
(a) Logical view



(b) Process view



(c) Development view



(d) Physical view

Figure 3: Recommended notions for the 4 main views of the 4+1 model [19]

+1 of the model, the so called *Scenarios*, are using a subset of elements and notions from the other views within use cases to display the interactions between objects and processes, thus not displayed in Figure 3. Kruchten is emphasizing that developing the views of the proposed model should be seen as an iterative process, making it possible to refine the model when needed. Customization of the model by omitting one or more phases can also be considered as not every project requires such a fine grained approach [19].

About the same time as the 4+1 Model was introduced, a group of authors, often cited as the "Gang of Four", published their famous book "Design Patterns: Elements of Reusable Object-Oriented Software" describing some of the most famous design patterns used within the field of software development.[16] Some of the patterns are explained within the next section.

3 Service Oriented Architecture and Micro-Services

Coming from the history of software architecture, this section describes an architecture from the present - the service oriented architecture. Furthermore, micro-services and their tenets are introduced.

3.1 Service Oriented Architecture

There can be found many publications about Service Oriented Architecture thus making it not easy to find a good definition. The following definition tries to describe the general purpose of Service Oriented Architecture:

> Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains [20].

This, and other definitions are lacking specific details and are not describing SOA in detail. According to Michael Stal, SOA is built around driving forces, using software patterns to satisfy them. He is describing the driving forces and the usage of software patterns by SOA in his publication "Using Architectural Patterns and Blueprints for Service-Oriented Architecture" [29]. The following list gives a brief overview of the driving forces that need to be balanced as described by the author:

- Distribution - different software running on different network nodes using a network protocol
- Heterogeneity - no control of remote service implementation details by developer, no knowledge of which services are called by whom in which contexts.
- Dynamics - decisions are subject of change at runtime
- Transparency - consumers of a services should not know about implementation details
- Process-orientation - compose multiple fine grained services to more coarse-grained ones as needed by clients

The following aspects are described in detail within the following paragraphs, according to [29], explaining their usage within the context of SOA.

- Interfaces and Contracts - clients access and understanding of services
- Communication - available communication styles, content and semantics
- State and activation - dealing with state information and stateless protocols, handling activation problems
- Service lookup and registration - make clients know about services, how clients locate services

- Processes and their implementations - service coordination and orchestration

3.1.1 Interfaces and Contracts

Loose coupling of clients and service implementation details can be accomplished by using the bridge pattern as it separates the interface from the concrete implementation [29]. Usually, when an interface/abstract class has multiple implementations, inheritance is used as non-abstract subclasses of the abstract class or interface are implementing it differently. A major drawback of this approach is that inheritance does not allow for much flexibility as abstractions and implementations are very hard to reuse independently and difficulties can arise when trying to modify or extend them. The Bridge pattern tries to solve this problem by creating an own class hierarchy for the abstractions/interfaces and a second hierarchy for the concrete implementations [16]. To allow distribution and to satisfy the need of transparency, the client side can use the proxy pattern, to hide the communication with the various services [29]. A proxy is acting as a placeholder/surrogate for another object to access/control it. Among this and other purposes, remote proxies can be used for encoding requests and sending it towards another object in a different address space. [16] This local proxy is also called "Ambassador" by Coplien et al. [15]. It is important to use interfaces together with an appropriate way of describing them to allow for heterogeneity. Providing a definition of the contract between clients and services using a domain specific language could include a description of the service functionalities and pre/post conditions, the services physical location, a semantic description of the service and further contextual information. A helpful pattern for providing this meta-information is the reflection pattern [29]. This pattern allows to change the behavior and structure of software systems during runtime by enabling the application to observe it's own state [26].

3.1.2 Communication

The communication between services and clients should preferably happen through dynamic messaging routes, using asynchronous messaging and various message queues. The handling of asynchronous reply message can be done using the observer pattern [29]. This pattern defines a one-to-many dependency between objects and notifies all objects that are part of this relationship about any changes in it's state so they are updated automatically [16].

3.1.3 State and activation

Maintaining state within applications decreases scalability and leads to higher coupling, thus SOA is aiming for statelessness, e.g. by implementing services using the singleton pattern, which ensures, that there is exactly one instance of the service class existing at runtime [16]. Additional patterns to increase scalability and decrease coupling are the activator pattern or the evictor pattern.

Furthermore, performance reasons are introducing the need to maintain at least some state information, thus the use of caching or other mechanisms can take place [29].

3.1.4 Service Lookup and Registration

The clients must locate services prior they are able to make use of them. A simple solution to this problem is provided by the Client-Dispatcher-Server pattern. It introduces the dispatcher, where the services can register themselves and clients can query the available services at the dispatcher, which acts as a repository of available services [29]. A even better solution is given by the lookup pattern, which allows for removing the location of the service repository and enables working with various kind of meta-information to find the needed services, like references or port numbers [17].

3.1.5 Processes and their implementations

Clients often need more than one service to accomplish their goal, thus composing multiple, fine-grained services together to a more coarse-grained service needs to take place. Furthermore, a coordinator pattern can be used to handle special requirements like the need for atomic transactions that need to be roll-backed in case of failure [29].

3.2 Introducing Micro-Services

When reading some vague definitions about Micro-Services, it can be hard to distinct them from the Service Oriented Architecture introduced by the previous section. According to Newman [21], Micro-Services are small and autonomous services, working together and having a single responsibility which they fulfill well. Some characteristics of Micro-Services are described by Shadija et al. as follows [27].

- Modularity of Services - changing one component and only need to re-deploy this single component thus moving away from multi-tier architectures
- Organization around business capability - Micro-Services as an application architectural style, each referring to a single business capability
- Products not Projects - shifting design focus towards the business/product
- Smart Endpoints and Dump Pipes - encapsulating all needed resources within the Micro-Services, enabling them via simple REST calls and using lightweight, asynchronous communication protocols
- Decentralized Data Management and Governance - management and distribution of data is not centralized and no centralized constraints regarding the data management must be followed

There are some common principles and characteristic occurring across various publications and Zimmermann [30] summarized them to the following list:

- Fine grained interfaces
- Business driven development practices like Domain Driven Design (DDD)
- Cloud native application design principles followed (IDEAL, twelve app factors of Heroku's method)
- Multiple computing paradigms in a polyglot programming and persistence strategy
- Usage of lightweight containers (e.g. Docker) for deployment
- Decentralized continuous delivery
- Lean and mostly automated configuration approach for DevOps

The next Figure, taken from [30], shows a 4+1 view, summarizing the characteristics of Micro-Services grouped by the authors Lewis and Fowler (LF-y), Newman (N-z) and by the principles listed by Zimmermann himself and stated in the listing above (T-x) cf. Figure 4).

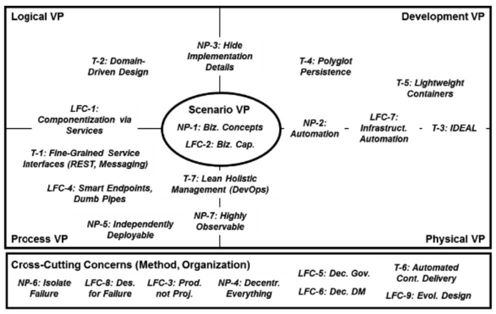
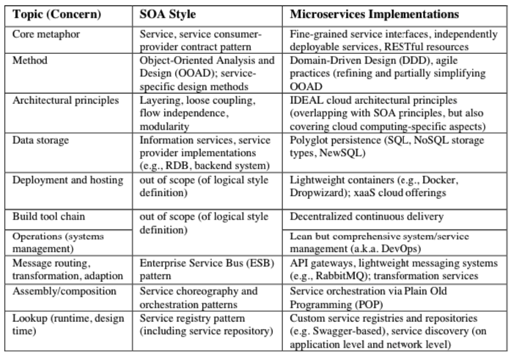

Logical VP
Development VP
NP-3: Hide Implementation Details
T-2: Domain-Driven Design
T-4: Polyglot Persistence
T-5: Lightweight Containers
LFC-1: Componentization via Services
Scenario VP
NP-1: Biz. Concepts
LFC-2: Biz. Cap.
NP-2: Automation
LFC-7: Infrastruct. Automation
T-3: IDEAL
T-1: Fine-Grained Service Interfaces (REST, Messaging)
T-7: Lean Holistic Management (DevOps)
LFC-4: Smart Endpoints, Dumb Pipes
NP-7: Highly Observable
NP-5: Independently Deployable
Process VP
Physical VP
Cross-Cutting Concerns (Method, Organization)
NP-6: Isolate Failure
LFC-8: Des. for Failure
LFC-3: Prod. not Proj.
NP-4: Decentr. Everything
LFC-5: Dec. Gov.
LFC-6: Dec. DM
T-6: Automated Cont. Delivery
LFC-9: Evol. Design

Figure 4: 4+1 View onto Microservices (Zimmermann 2007)

The usage of new, lightweight technologies like Docker container or ZeroMQ and the shift away from multi-tier architectures, together with the encapsulation of needed resources into the services, do evolve Micro-Services from the past SOA applications. Again, taken from [30], the last figure of this section is displaying the relationships and differences of SOA and Micro-Services in a compact summary cf. Figure 5).



Topic (Concern)	SOA Style	Microservices Implementations
Core metaphor	Service, service consumer-provider contract pattern	Fine-grained service interfaces, independently deployable services, RESTful resources
Method	Object-Oriented Analysis and Design (OOAD); service-specific design methods	Domain-Driven Design (DDD), agile practices (refining and partially simplifying OOAD
Architectural principles	Layering, loose coupling, flow independence, modularity	IDEAL cloud architectural principles (overlapping with SOA principles, but also covering cloud computing-specific aspects)
Data storage	Information services, service provider implementations (e.g., RDB, backend system)	Polyglot persistence (SQL, NoSQL storage types, NewSQL)
Deployment and hosting	out of scope (of logical style definition)	Lightweight containers (e.g., Docker, Dropwizard); xaaS cloud offerings
Build tool chain	out of scope (of logical style definition)	Decentralized continuous delivery
Operations (systems management)		Lean but comprehensive system/service management (a.k.a. DevOps)
Message routing, transformation, adaption	Enterprise Service Bus (ESB) pattern	API gateways, lightweight messaging systems (e.g., RabbitMQ); transformation services
Assembly/composition	Service choreography and orchestration patterns	Service orchestration via Plain Old Programming (POP)
Lookup (runtime, design time)	Service registry pattern (including service repository)	Custom service registries and repositories (e.g. Swagger-based), service discovery (on application level and network level)

Figure 5: Summary of relationships between SOA and Micro-Services (Zimmermann 2007)

4 videoOCR - OCR Source Code Extraction and Visualization of Pupil Data

4.1 Motivation

The implementation of videoOCR is aiming to provide a tool for extracting and locating source code within video recordings and analyzing as well as visualizing data, gathered by eye-tracking experiments.Goal/Purpose of this thesis is to map eye-gaze positions,acquired with ET glasses to source code in order to extract AoIs and fixations and reveal viewing patterns. While solving programming tasks, the participants were wearing eye tracking glasses capturing their eye movements. The experiment is aiming to compare different architectural styles, mainly in terms of comprehension, and to provide solutions to support developers when confronted with unknown source code and software architectures they are not familiar with. The practical part of this thesis tries to implement a solution to extract the source code from the participants recordings, which includes the paths of their eye movements, identifying longer fixations of the eyes and to visualize their navigation through the source code. Furthermore this thesis aims to provide an implementation to automatically extract the source code using optical character recognition (OCR) and to map the eye movements onto it, thus providing a visualization enriched with metadata for further analysis. The experiment consisted of multiple tasks and one task was chosen to be part of the analysis. Within this task, the participants had to find a certain class within the source code, specifically a class that checks the validity of a payment card.

4.2 Related Work

Generally speaking, OCR is a widely used technique to extract text from images and there are some publications concentrating on extracting source code written in various programming languages. In [10], Alahmadi et al. are focusing on locating source code from programming tutorial videos using deep learning but without processing OCR. They analyzed 150 videos and the extracted frames had to be manually annotated by the developers with the bounding boxes of the source code to use it as the ground truth. The authors stated that they reached about 92% accuracy in predicting the precise location of code fragments within the video when compared to the ground truth. More focused on the extraction of code using OCR is the work of Ponzanelli et al. [25]. Within this paper, the authors introduced CodeTube, a tool that indexes video tutorials using the Tesseract OCR engine and further delivers watch suggestions to relevant video fragments related to the users search query.

The implementation of videoOCR is combining the localization and extraction of source code from recorded videos together in a single application but without the usage of machine learning. Instead, videoOCR is using the data gathered from external tools like the Pupil Core headset together with the Pupil Player and other data, including trace files, showing which files where opened when. This input is not only helping to localize and identify the visible source code in the video, it is also used to analyze the eye gaze movements of the user.

4.3 Method

Developing Micro-Services is different from developing monolithic applications. Balalaie et al. identified some interesting points in their experience report [11], when migrating to Micro-Services, whereas the following can be the most challenging for developers:

- Development - local deployment of all dependent Micro-Services is needed, thus creating more complexity, especially for novice developers
- The Need for Skilled Developers - introducing Micro-Services means introducing a lot of additional technologies (load-balancers, messaging, service registry etc.), thus requiring additional skill-sets
- Polyglot Persistence and Programming - Micro-Services are encouraging polyglot development, which can lead to more complexity.

The authors conclude that, before migration to Micro-Services, the challenges arising with this decision should be considered carefully as Micro-Services are not always the silver bullet solution for existing architectural problems [11]. The usage of eye tracking can help finding solutions to this challenges and similar eye tracking experiments were conducted before at the University of Vienna to gain insights in the cognitive load, like in [28], where the authors analyzed eye movements of subjects when confronted with unknown language features. Executing eye tracking experiments to analyze the difficulties arising for the developer when working with Micro-Services can be seen as opportunity to gain better insights in the cognitive challenges for the individual subjects as it provides a cost-efficient and reliable method to measure cognitive processes of developers during work. The eye tracking data was recorded with a Pupil Labs Core headset [2], equipped with two cameras (200Hz eye camera, frontal world camera with a resolution of 1280x720 pixels). The recording was done with the pupil capture software and consists of eye-movements, fixation detection, surface mapping and various other data that shall not be described in detail here. As stimuli, the participants had to solve various programming tasks within an e-commerce application, implemented twice, using two different architectural styles, once being composed of Micro-Services and once being implemented using a monolithic architecture. A screen recording also took place, capturing the IDE window. The programming tasks which the participants had to work on within the experiment, are based on two GitHub repositories, whereas [4] is the Micro-Service implementation and [6] the monolithic implementation of the same e-commerce application. The thesis focuses on analyzing a single task where the subjects had to localize a specific class, responsible for checking the validity of payment cards. This requires cutting the screen recording to only

[2]https://pupil-labs.com

contain the specified task as well as exporting only the eye tracking data referring to this time frame using the Pupil Player software. The exported data included various files from which this analysis used the eye fixation and gaze distribution data across the surfaces. The surfaces must be specified prior to the export within the Pupil Player software and are defining the different parts of the recording (navigation, code editor, chat, etc.). Furthermore, a trace file is generated while capturing the screen, which provides information about the opened files together with timestamps describing when the corresponding files were opened (also to be edited manually to only contain the files used by the analyzed task). The implementation is extracting frames as JPEG images at a rate of 60 fps and processing every image with OCR to extract the text (source code) from the image frames. The extracted text is then compared with the parsed original source code as ground truth to find matching classes and methods that occurred in the recording. To allow for a certain error rate in OCR, the matching is done using 2 text similarity algorithms. The first algorithm used is the Jaccard Similarity, calculating the intersection of words divided by the union as explained in [5]. A sample application can be found in [22]. The second algorithm used is the Jaro Winkler similarity algorithm, indicating the percentage of matched characters between two character sequences [14]. A Jaccard index of 0.96 combined with a 0.95 index result by Jaro Winkler, led to reliable results. Smaller numbers make it impossible to identify certain methods only differing in returning a Set instead of a List, for example. Performing further algorithms, the implementation generates visualizations for eye fixations, gaze distributions and a diagram showing the navigation of the subjects through classes and methods.

4.4 Solution Requirements

This section is briefly describing the features of the application, thus providing the Definition of Done for the application. Furthermore, an evaluation of the features was done if applicable.

4.4.1 Frame extraction

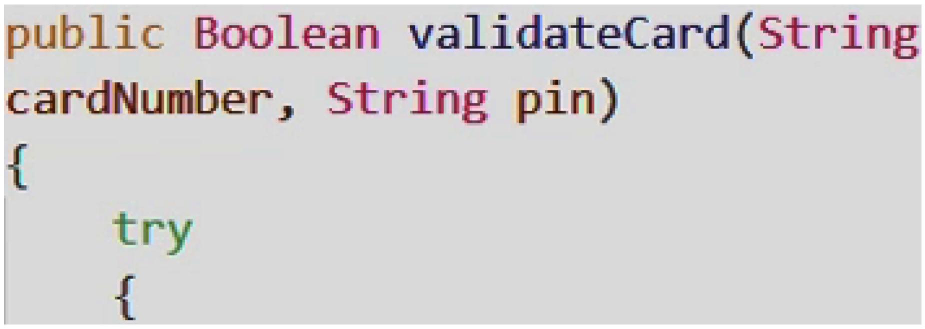
Functional Requirements

- The system is being capable of extracting frames from an input video.
- The user can select from multiple input files.
- The user can add video files to the selection list via choosing local files and uploading them.
- The user can define the dimensions of the input area.

Non-Functional Requirements

- The frame extraction process must accept input video files encoded in M4V (.m4v) format.
- The frame extraction generates an image file for each processed frame.
- Each generated image file can be reassigned to the corresponding frame afterwards, either manually or programmatic.
- The resulting image quality is good enough to be processed by the OCR engine
- The file size for each image should not exceed 1 MB
- The resulting output images are stored in separate folders, named corresponding to their given input video file names.

Evaluation Criteria
The frame extraction process can be evaluated by analyzing the quality and the file size of the extracted images. The evaluation was performed two times, first extracting frames without compression using the TIFF image format, followed by a second run, extracting frames with compression using the JPEG image format, both resulting in images with a dimension of 532 x 755 pixel.



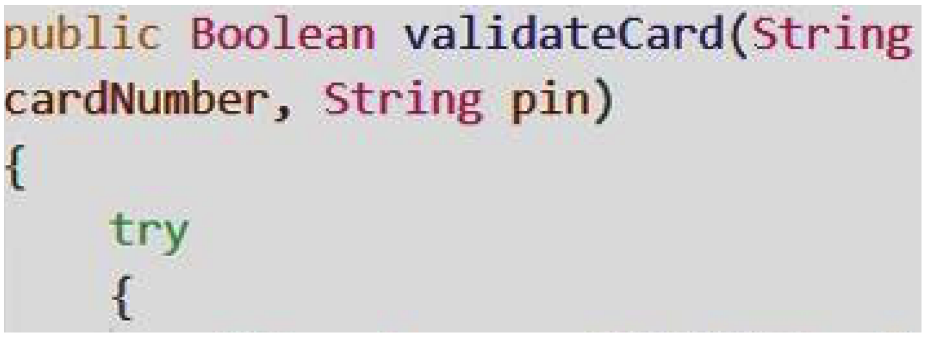
```
public Boolean validateCard(String
cardNumber, String pin)
{
    try
    {
```

Figure 6: TIFF Image Snippet, Original Size: 1.2 MB (1,205,924 Byte)

The differences regarding file sizes between uncompressed TIFF images and compressed JPEG images were remarkable. Using uncompressed images would only be acceptable when leading to significantly better OCR results, although there were huge differences between uncompressed and compressed files in image quality. (cf. Figure 6 and Figure 7 above). A deeper analysis regarding the quality differences is done within the evaluation section of the OCR extraction process.

4.4.2 OCR Source Code Extraction

Functional Requirements



```
public Boolean validateCard(String
cardNumber, String pin)
{
    try
    {
```

Figure 7: JPEG Image Snippet, Original Size: 55 KB (55,015 Byte)

- The system processes the given input frames and extracts all recognized text from it.
- The user can select which frames to process from the list of uploaded video files.

Non-Functional Requirements

- The extraction process accepts 1-n JPEG images, each representing a single frame, as input.
- The input is taken from the output folder of the frame extraction, corresponding to the selected video file name.
- The results of each frame are collected, generating a single JSON file as output.
- The text recognition is done line by line for each frame.
- Each line is accompanied by additional metadata containing at least the line-coordinates and dimensions within the frame and the frame number of the corresponding frame.
- The output file is stored in a separate folder for each input video file
- The OCR process is deterministic, producing the same output given the same input, ceteris paribus.
- The accuracy of the generated output matches the criteria described in the evaluation criteria section below.

Evaluation Criteria
The projects overall quality is highly dependent on the accuracy of the OCR process, thus requiring the evaluation process to be based on reliable evaluation

criteria. Generally speaking, the successful completion of the evaluation process is indicated by a predefined level of accuracy that must be reached after analyzing the output of the OCR process. The evaluation was done using a tool developed by Rafael C. Carrasco at the University of Alicante as described in [13]. The tool is generating a report containing the Character Error Rate (CER) and the Word Error Rate (WER), respectively measuring the accuracy on character- (CER) and word-level (WER). The input parameters needed for the computation, besides the count of characters (C) or words (N), are the number of (S)ubstitutions, (D)eletions and (I)nsertions compared to the ground truth (cf. Figure 8). The evaluated ground truth and the corresponding hOCR

$$CER_{normalized} = \frac{S+D+I}{S+D+I+C}$$

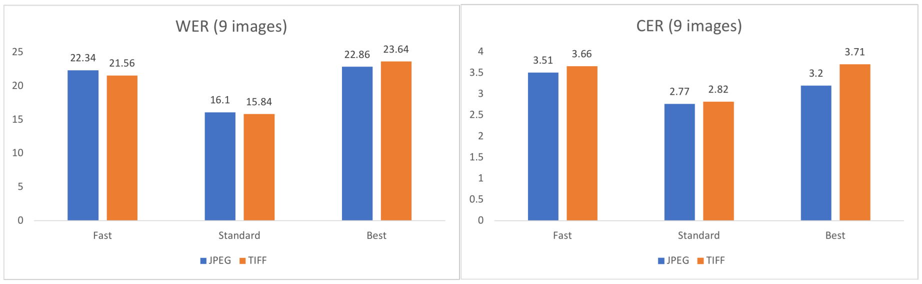
(a) CER Formula

$$WER = \frac{S_w + D_w + I_w}{N_w}$$

(b) WER Formula

Figure 8: Calculation of Character- and Word-level error rates [1]

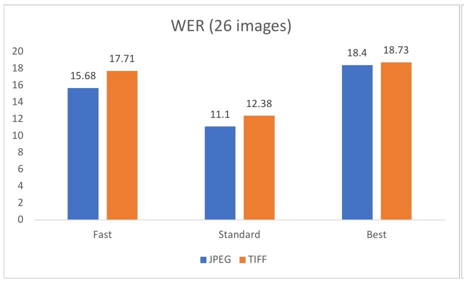
result were manually selected from a participant's recording and captured parts of 3 classes. For each class, 3 sample frames were selected - 9 frames in total. After manually transcribing each frame to a corresponding plain text file, the OCR process was executed, resulting in 9 hOCR files. The OCR process was executed multiple times, each time using a different Tesseract .traineddata file, covering the best, standard and the fast variant of the pre-trained English language data set used by the Tesseract OCR engine to produce the hOCR result files [9]. After gathering the input, a programmatic calculation of the error rates was done using the evaluation tool and resulting in HTML reports Figure 22). The output of the error rate computation was quite surprising, as best results



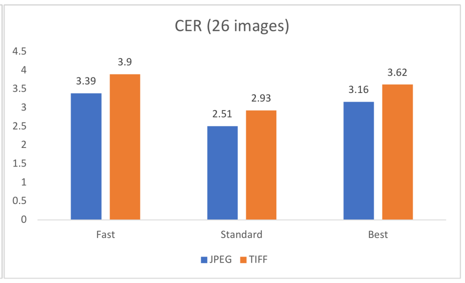
(a) Word Error Rates (b) Character Error Rates

Figure 9: Evaluation Results First Run

for the best traineddata langauage data set and correspondingly the worst for the fast traineddata were expected but the standard data set created the best results (cf. Figure 9). Furthermore, executing the OCR process with uncompressed TIFF images did not improve the overall results for each data set. After analyzing the result, a second evaluation was done, using a larger input data set with 26 images, as larger input samples are leading to more reliable output although the sample size was constrained by the manual image transcription. Affirming the results of the first run, the Tesseract OCR engine should be used



(a) Word Error Rates



(b) Character Error Rates

Figure 10: Evaluation Results - Second Run

with JPEG images as input and configured to use the standard English language data set. (cf. Figure 10).

4.4.3 Mapping Extracted Code to Ground Truth

Functional Requirements

- The feature is reading the output of the OCR extraction and matches the extracted lines with the original source code.
- The project path of the ground truth is scanned and the found java code is parsed to be used by the matching algorithm.
- The parser only takes classes into consideration that were visited within the chosen input video, files unrelated to the video are being ignored.
- The extracted lines of the OCR process are compared to the original source code using text similarity algorithms.
- The algorithm can identify which methods of the visited classes occur in the frames.
- The algorithm is mapping the matched methods correctly to the classes thy belong to, meaning the mapping can distinguish methods of different classes although their method signature may be exactly the same.
- One resulting output is a list of visited methods together with the corresponding classes as well as the duration of the frame they were occurring at, accompanied with the coordinates on which they are appearing within the frame of the given duration.

- Additionally, a second output list is being generated, again containing the visited classes/methods. It further contains the accumulated total duration of each methods thus providing a list of methods sorted by the total duration they were visible within the original video.

Non-Functional Requirements

- The input data to be compared with the ground truth is read directly from the JSON file previously generated by the OCR source code extraction process.
- The output is resulting in two JSON formatted files.
- The used algorithms must produce deterministic results for the same input.
- The input data for the ground truth is derived from the video name.
- The parsing of the ground truth must produce deterministic results for the same input.
- The accuracy of the result is determined by the number of methods identified divided through the number of methods occurring in the video.
- The average accuracy of the result should be above 70%.

Evaluation Criteria Evaluating a sample recording, the accuracy of the result was 75% as 9 out of 12 methods and all visited classes were identified correctly. The not recognized methods didn't fit the editor window and therefore a line break occurred somewhere in the middle of the declaration, making it impossible to recognize them as the recognition is done line by line.

4.4.4 Process - Analyze - Visualize Eye-Tracking Data

Functional Requirements

- The presentation and visualization of the results are displayed to the user within the browser.
- The user can select a distinct recording from a list. The generated output is based on this selection.
- The participant's navigation within classes and from class to class is visualized by a PlantUML diagram.
- The participant's eye-gaze movements are visualized by the path corresponding to their eye-gaze fixations.
- The total eye-gaze duration on each, distinct surface (code, navigation, other) is visualized by a pie chart.
- The total eye-gaze duration, visualized by the pie chart is measured in percentage of the total video duration.

Non-Functional Requirements

- The animations run smoothly and are not slowing down the browser
- The input data used for visualizing the class navigation and total class/-method visibility is taken from the generated output of the mapping process.
- The input data needed for the other visualizations (e.g. eye-gaze movements) are taken from the exported output data of the Pupil Player.
- The format of the input data taken from the mapping process JSON
- The format of the input data taken from the Pupil Player export is CSV.

4.5 Choosing an OCR Engine

The importance of choosing the right OCR Engine for the implementation is an important task, as choosing an insufficient engine can have an impact on the quality of both the implementation and the produced results or lead to other problems, like high complexity or compatibility issues, caused by the usage of outdated technologies and dependencies. The first attempt to find OCR engines to work with, only based on the German Wikipedia page about OCR [8] and on a listing from a Github Repository about OCR projects that it's author considers to be 'awesome' [3], already led to an overwhelming list of over 40 different solutions to choose from. The following table lists a random choice of some of the found OCR-engines (cf. Table 1): Given the sheer number of existing OCR

ABBY Cloud OCR	ArgusScript	attention-ocr	BIT-Alpha
Calamari	Clara OCR	CuneiForm	dpScreenOCR
EasyOCR	Eye	FineReader	FormPro
GOCR	Google Cloud Vision	GT Text	hebOCR
KADMOS	kognition	kraken	NSOCR
OCR-D	OCR.space Online OCR	OCR4all	Ocrad
OCRchie	ocre	OCRFeeder	Tesseract

Table 1: Examples of OCR engines found online

engines, the attempt to identifying possible candidates that can be used realizing the implementation of the project by just searching for available applications is not sufficient. There is a need for a more structured approach for choosing the right OCR engines as there are not only too many of them available, some of them simply don't offer API support or have other constraints. HebOCR, for instance, is specifically designed for recognizing Hebrew and therefore not capable for source code recognition [2]. Prior to the continuation of the decision process, there must be some constraints a OCR engine must fulfill to be taken into consideration.

4.5.1 Decision Criteria

The problems identified on the first approach to choose a OCR engine to work with, are implying the need for objective decision criteria to support the decision process. The engine must offer support using it with Java API calls as this is a desired feature of the implementation. Furthermore it should be capable to process text written in English and German as the captured source code is using English as main language and there may be some occurrences of German words. The usage of the engine has to be free of charge or offer a free usage plan as the project is non-commercial and planned to be used only within scientific contexts. The statements mentioned before are pretty straightforward and can be answered true or false and make absolute decision criteria which must be fulfilled. The engine must produce results accurate enough to allow the mapping of the participants eye gazes to the exact location within the source code. Frames captured during scrolling through source code, can lead to more inaccurate text recognition. These scroll events should be handled properly by the engine, leading to results with acceptable accuracy. The reflections about what constraints an engine must fulfill lead to the following list of decision criteria:

- Absolute criteria
  1. Supporting usage through Java API calls
  2. English and German language support
  3. Using the OCR engine must be free of charge
- Relative criteria
  1. Sufficient accuracy
  2. Acceptable performance
  3. Easy usage
  4. Easy installation

The items categorized under absolute criteria must be fulfilled all together since lacking support for using Java API's and the engine not being able to process the source code mainly written in English does not match the implicit requirements needed for the use case of this thesis. Furthermore, there should be no obligation to pay for the engine as this project is intended to be used in an academic context and therefore should not consist of any monetary constraints whatsoever. The relative criteria are not absolute, meaning they are not broken down to yes/no decisions. Implementing a solution to test the various OCR engines is not a trivial task, as they often require third-party libraries and dependencies (e.g. Leptonica is required by Tesseract) and only a few of them are providing Java API's that are sufficing the need of this application. The Tesseract OCR engine is fulfilling all absolute criteria and due to the fact that it is widely used, well documented and providing easy usage together with Tess4J as wrapping library for Java, no further tests were taken to find another OCR engine.

4.6 Implementation

4.6.1 Technologies and Frameworks

Before starting to program, decisions regarding the used technologies and frameworks had to be done. The following list is a summary of the ones used within this project:

- Java 11 (Programming Language)
- HTML + JavaScript + CSS (Webcontent)
- Maven 3.5.4 (Build Management)
- Spring Boot 2.4.4 (DI, REST, Web MVC Backend)
- Thymeleaf (Web MVC Frontend)
- D3.js (JavaScript Visualization Library)
- Jaffree 2021.08.04 (Frame extraction)
- Tess4j 4.5.0 (Optical Character Recognition)
- JavaParser 3.20.2 (Source Code Parser)
- Plantuml-core 1.0.40 (PlantUML Generation)
- Graphviz-java 0.18.1 (Plant UML Generation)
- Json Simple 1.1.1 (JSON)
- Junit 4.13.2 (Unit Testing)
- Apache Commons Lang 3 (String Helper Utilities, Pair Implementation)
- Apache Commons Text (Jaccard-, and Jaro Winkler Similarity Algorithm Implementation)
- Docker (Containerization)

The backend was written in Java 11, using Maven 3.54 to manage the builds and necessary dependencies, due to the fact that I am using these technologies on a daily basis and therefore are most familiar with and using Gradle as an alternative to Maven is not making any difference for the project. Using Python or C++ as programming language could have been taken into consideration as there exist native frameworks for FFMpeg and Tesseract for these languages which offer a lot more functionality than the wrapper frameworks for JAVA. But the lack of knowledge about the ecosystem and to write good code in these languages, especially in C++, lead to using JAVA as the only alternative. The web content is being served by Spring Boot and it's MVC functionality together with Thymeleaf in the frontend. Besides Thymeleaf, there were a lot of other frameworks to choose from like Angular, React, Vue.js and many more, the reason for choosing Thymeleaf was mainly it's simplicity and seamless integration with Spring Boot. Thymeleaf can be used without configuration and supplements ordinary HTML pages with extra functionality to serve web content offered by Spring Boot. The mentioned alternative frameworks offered much more features but as a consequence of that, needed much more configuration and time to learn until someone can see any progress. Furthermore, there is no need for a sophisticated frontend as the visualization is done by another JavaScript framework - D3.js. Generally speaking, D3.js is a library for manipulating the DOM based on data and primarily used for visualization of data, e.g in the form of bar charts, force-directed graphs and various other diagrams or charts. The JavaScript ecosystem offers many libraries for this purpose, like the already mentioned React and Vue, as well as GraphViz, HighCharts and so on. D3.js was used because it offered the largest community and the best documentation, although it has a very steep learning curve and a few other, time-consuming problems which are discussed further within the next section where the implementation process is discussed in detail. The extraction of frames as JPEG files from the screen caputure is done by Jaffree (JAva FFmpeg and FFprobe FREE command line wrapper). As implied by the name, Jaffree provides a simple wrapper around the FFmpeg command line tool, thus offering a simple API for executing FFmpeg and FFprobe commands. FFmpeg as a command line tool is widely used for processing video streams and the Jaffree repository contained an example of the usage for frame extraction. Therefore this example was used as a template for the project implementation, only a few adoptions were necessary. Because of the simple usage and good documentation there was no searching for alternatives done. The choice of Tesseract as OCR engine narrowed the library candidates to be used for OCR to Tess4J. This is the most used and most popular Java wrapper for Tesseract. Good documentation and simplicity made further research obsolete. JavaParser is a library for Java source code parsing. The use case within this project is the parsing of the ground truth - the source code of the microservice sales app and it's monolithic counterpart to find matching classes and methods within the extracted source code of the screen capture video. Two algorithms came in handy for finding matching class and method candidates, the Jaccard similarity algorithm combined with the Jaro Winkler similarity algorithm. Both of them were implemented by the Apache Common Text library. PlantUML is used together with Graph-viz to create a visualization of the navigation between classes and methods by the participant in the style of UML diagrams. The application can be deployed within a Docker container and a dockerfile is included in the source code. The next section describes the implementation process together with the problems that occured during the process as well as other findings.

4.7 Implementation process - Findings and Problems

This section is going into more detail about the most important milestones of the programming process and what kind of problems occurred during that.

4.7.1 Frame extraction

The extraction of frames is done by Jaffree, which is a wrapper around the command line tools FFMpeg and FFProbe. These tools have to be installed on the system where the project's application is running on. The following listings are showing the frame extraction with FFmpeg and the custom implementation of a frame consumer.

Listing 1: Usage of Jaffrees FFmpeg Class to extract Frames

```
FFmpeg.atPath()
    .addInput(UrlInput
        .fromPath(videoPath)).setFilter(StreamType.VIDEO, "negate")
    .setProgressListener(progress -> {
        frameCount.set(progress.getFrame());
        currentTimeMillis.set(progress.getTimeMillis());
        double percents = 100. * currentTimeMillis.get() / duration.get();
        final BigDecimal percentage = BigDecimal.valueOf(percents).setScale(2,
 RoundingMode.HALF_UP);
            LOG.info("Frame extraction progress:y {} %", percentage);
        })
    .addOutput(FrameOutput
        .withConsumer(new VideoFrameConsumer(framesPath, currentTimeMillis,
 boundingBox))
        .setFrameRate(60)
        .disableStream(StreamType.AUDIO)
        .disableStream(StreamType.SUBTITLE)
        .disableStream(StreamType.DATA))
    .execute();
```

Listing 2: Implementation of FrameConsumer (omitting Members)

```
public class VideoFrameConsumer implements FrameConsumer
{
   @Override
    public void consume(Frame frame)
    {
        // End of Stream
        if (frame == null)
        {
            return;
        }

        final BufferedImage subImage = boundingBox != null ?
            ImageHelper.getSubImage(frame.getImage(), boundingBox.x, boundingBox.y,
 boundingBox.width,
                boundingBox.height) : frame.getImage();

        final String filename = duration.get()
            + ".jpg";
        final Path output = pathToDstDir.resolve(filename);

        try
        {
            ImageIO.write(subImage, "jpg", output.toFile());
23            }
24            catch (IOException e)
25            {
26                e.printStackTrace();
27            }
28
29        }
```

Before writing the consumed frame as JPEG file to the disk, the consumer scales it to the right dimensions as seen on the corresponding lines 12+13 of listing 2. The first implementation of the frame consumer was just taking the image without any modifications. Furthermore, a filter is applied to the input video as seen at line 3 of listing 1, inverting the colors. The adoption was necessary because performing OCR without it, was resulting in unusable output.

4.7.2 Performing OCR

Performing optical character recognition on the frames was done with Tess4J. Tess4J requires Tesseract to be installed on the system analogues to Jaffree, which required FFMpeg to be installed. As mentioned within the frame extraction section, extracting frames from the input video without inverting colors and scaling the image led to bad results. This is because the participants of the experiment all used a black theme within the IDE as well as different surface sizes (code, navigation, chat window). Tesseract works best with black fonts on white background, but just inverting the colors seemed to be a good compromise. After applying these modification, significantly better results were produced as output.

Listing 3: Implementation of Performing OCR (lines omitted)

```
...
@Override
public void extractTextFromFramesToJSON(String fileName)
{
...
    final Stream<Path> parallelStream = pathStream.parallel();
    parallelStream.forEach(framePath -> {
        final Tesseract tesseract1 = getTesseractInstance(hocr);
        try
        {
            longAdder.decrement();
            if (longAdder.sum() % 10 == 0)
            {
                LOG.info("extracted text from {} frames of {}", longAdder.sum(), count);
            }
            wordContainerList.add(retrieveWordContainerForFrame(tesseract1, framePath.toFile()));

        }
        catch (TesseractException e)
        {
            LOG.error("exception occurred during extracting linesy of frame: {}", framePath, e);
        }
        });
        writeWordContainerListToJSON(wordContainerList,
    pathContainer.getExtractedLinesPath());
}
...
private WordContainer retrieveWordContainerForFrame(Tesseract1 tesseract1, File
    frameFile) throws TesseractException
    {
        final OCRResult documentsWithResults;
        final String absoluteFileString = frameFile.getAbsoluteFile().toString();
        final String duration = StringUtils.getDigits(frameFile.getName());
        documentsWithResults =
            tesseract1.createDocumentsWithResults(absoluteFileString, absoluteFileString,
                Collections.singletonList(ITesseract.RenderedFormat.HOCR),
                ITessAPI.TessPageIteratorLevel.RIL_TEXTLINE);

        final WordContainer wordContainer = new WordContainer();
        if (StringUtils.isNoneBlank(duration))
        {
            wordContainer.setDuration(Long.parseLong(duration));
        }
        else
        {
            wordContainer.setDuration(11L);
        }
        wordContainer.setWordList(documentsWithResults.getWords());

        return wordContainer;
    }
```

The resulting WordContainer list gets written to a JSON file thus making it possible to process it independently later on. The word container class acts as a custom container for the result produced by Tess4J. It contains the current duration timestamp of the frame as well as the recognized lines which are encapsulated within Tess4J's Word class. A Tess4J word contains the recognized string together with it's rectangle bounding box and the recognizing confidence. Note that the term Word has got a different meaning than implied. Defined by the TessPageIteratorLevel, Tesseract will generate 1 Word containing the result of the recognition by different scopes, e.g. RIL TEXTLINE defines a line within a paragraph as the scope for a word whereas RIL WORD generates a word for each word within a textline. The first tryouts were done using the word level and later changed to the text line level because all other options were not applicable within the context of this application.

4.7.3 Parsing the Ground Truth and Matching Classes/Methods

To find out which methods and classes a participant visited, the extracted lines must be compared and matched with the source code they worked on, the so called ground truth. The Java Parser library provides the functionality to parse Java source code and generating the output needed for the matching. The next listing shows some implementation parts of getting the class and method names, based on the visited file names, taken from the editor trace file contained in the experiment data of each participant.

Listing 4: Parsing the Ground Truth (lines omitted)

```java
 private List<ClassContainer>
    parseCodeFromGroundTruthAndBuildMatchingClassContainerList(
        PathContainer pathContainer) throws IOException
    {
        LOG.info("Finding class matching candidates and creating class/methods map for
    method name similarity search");

        final Map<String, List<String>> parsedMethodNamesPerClass =
    buildParsedClassMethodMap(pathContainer);
        final List<ClassContainer> visitedClassesFromTraceEditor =
            readVisitedClassesFromEditorTraceFiles(pathContainer);

        return addParsedMethodsToVisitedClasses(parsedMethodNamesPerClass,
    visitedClassesFromTraceEditor);
    }
```

The trace editor files had to be edited manually prior because they contained the trace of the whole recordings, but the recordings were cut to only contain a single programming task. The parsing uses the visitor pattern and relies on a custom implementation of a GenericVisitorAdaptor provided by the Java Parser library. This adapter returns a map of all methods of the classes within the ground truth.

Listing 5: Visitor pattern used to parse the classes (lines omitted)

```java
public class ClassMethodVisitorAdapter extends GenericVisitorAdapter<Map<String,
    List<String>>, Object>
{
    @Override
    public Map<String, List<String>> visit(ClassOrInterfaceDeclaration n, Object arg)
    {
        final Map<String, List<String>> classMethodMap = new HashMap<>();
        classMethodMap.putIfAbsent(n.getFullyQualifiedName().orElse(null),
    n.getMethods()
                .stream()
                .map(methodDeclaration ->
    methodDeclaration.getDeclarationAsString(true, true, true))
                . collect (
                    Collectors . toList ()));

        return classMethodMap;
    }
}
```

After that, the editor trace files are loaded and a new list is created, containing only classes which where visited by the participant.

Listing 6: Creation of a List of Class Container only with visited classes by participant (lines omitted)

```java
private List<ClassContainer> addParsedMethodsToVisitedClasses(Map<String,
    List<String>> parsedMethodNamesPerClass, List<ClassContainer>
    visitedClassesFromTraceEditor)
```

```
{
    parsedMethodNamesPerClass.forEach((className, methodNames) ->
        methodNames.forEach(methodName ->
            visitedClassesFromTraceEditor.forEach(classContainer ->
            {
                if (StringUtils.equals(className,
     classContainer.getFullyQualifiedClassName()))
                {
                    
     classContainer.setMethodNameList(parsedMethodNamesPerClass.get(className));
                }
            }
        )
    )
);
return visitedClassesFromTraceEditor;
}
```

4.7.4 Visualization of Data

The main result of the visualization process is a PlantUML diagram, describing the time-dependent navigation through methods and classes. The next listing shows an excerpt of the PlantUML diagram creation.

Listing 7: Creation of PlantUML diagram (lines omitted)

```
    private String createPlantUMLString(List<ClassContainer> visitedClassContainerList,
        List<MethodContainer> totalDurationMethodList,
        List<MethodContainer> matchedMethodList) throws IOException
    {
        totalDurationMethodList.forEach(methodContainer ->
     visitedClassContainerList.forEach(classContainer -> {
            if (StringUtils.equals(methodContainer.getClassName(),
     classContainer.getFullyQualifiedClassName()))
            {
                classContainer.getMethodContainerList().add(methodContainer);
            }
        }));

        final StringBuilder stringBuilder = new StringBuilder();

        stringBuilder.append("@startuml")
            .append('\n')
            .append("hide empty members")
            .append('\n')
            .append("skinparam roundcorner 20")
            .append('\n')
            .append("skinparam linetype ortho")
            .append('\n')
            .append("skinparam nodesep 200")
            .append('\n')
            .append("skinparam ranksep 200")
            .append('\n');

        buildPlantUmlClasses(visitedClassContainerList, totalDurationMethodList,
     stringBuilder);
```

```
        buildPlantUmlClassLinks(visitedClassContainerList, stringBuilder);
        buildPlantUmlMethodLinks(matchedMethodList, stringBuilder);
        stringBuilder.append("@enduml");

        return stringBuilder.toString();
    }
     ....

    private void buildPlantUmlClassLinks(List<ClassContainer>
      visitedClassContainerList, StringBuilder stringBuilder)
    {
        String previousClass = null;


      visitedClassContainerList.sort(Comparator.comparingLong(ClassContainer::getOpenedFrom));

        for (ClassContainer classContainer : visitedClassContainerList)
        {
            final String currentClass =
      StringUtils.remove(classContainer.getFullyQualifiedClassName(), ".");

            if (previousClass != null && !StringUtils.equals(currentClass, previousClass))
            {
                stringBuilder.append(previousClass)
                    .append("..>")
                    .append(currentClass)
                    .append(" :")
                    .append(classContainer.getClosedAt())
                    .append('\n');
            }

            previousClass = currentClass;
        }
    }
```

The PlantUML string is converted into a .svg file as well as into a .txt file. Both of them can be downloaded. The plain text file can then further be processed or displayed by other applications using PlantUML. For visualizing the participants eye-gaze movements and their distributions across the various surfaces (code, navigation, chat...), Javascript along with D3.js was used. The visualization requires the manual uplaod of the csv files containing the eye-gaze positions and the normalized gaze distributions. The following listing partly shows the implementation of the gaze distribution visualization.

Listing 8: Creation of Gaze Distribution diagram (lines omitted)

```

function visualizePieChart(videoName) {
    var s = "/extracted-dir/" + videoName +
     "/vizData/normal_surface_gaze_distribution.csv";

    d3.csv(s, function (error, data) {
        if (error) {
            throw error;
        }
        var arc = g.selectAll(".arc")
```

```
            .data(pie(data))
            .enter().append("g")
            .attr("class", "arc");

        arc.append("path")
            .attr("d", path)
            .attr("fill", function (d) {
                return color(d.data.surface_name);
            }).on("mouseover", onMouseOver) //Add listener for the mouseover event
            .on("mouseout", onMouseOut)   //Add listener for the mouseout event

        console.log(arc)

        arc.append("text")
            .attr("transform", function (d) {
                return "translate(" + label.centroid(d) + ")";
            })
            .text(function (d) {
                return d.data.surface_name;
            });
    });
}

svg.append("g")
    .attr("transform", "translate(" + (width / 2 - 120) + "," + 20 + ")")
    .append("text")
    .text("Gaze counts per Surface")
    .attr("class", "title")
```

4.8 Results

The results of the evaluation are three different visualizations, an eye gaze distribution diagram, a diagram which animates the eye gaze fixations and a customized UML diagram, taking a deeper look on how the subjects navigated through the source code.

4.8.1 PlantUML Class Diagram

Structure
The generated PlantUML diagram contains classes with methods and links from classes to classes as well as links from methods to methods. The method links within the same class are labeled with how many times this transition occurred (e.g x2 for two transitions). The class links are labeled with the timestamp of the transition. Furthermore, the classes are colored with a gradient, showing the time-frame on which they were visited. Looking at (cf. Table 2), a class first opened at 21s and transitioning to another class at 75s means that the class is colored with a vertical gradient, beginning with a yellow colorization and ending colored in red. The method names appear with different font sizes, regarding the total visibility duration and scaled between a minimum font size of 8 and a maximum font size of 40.

Color	Start	End
White	0s	0s
Yellow	0.01s	29.99s
Orange	30s	59.99s
Red	60s	89.99s
Green	90s	119.99s
Pink	120s	149.99s
Purple	150s	179.99s
Blue	180s	-

Table 2: Color Gradient for Classes

Discussion
Looking at the first three classes visible (cf. Figure 11), two transition between classes and 2 method transitions can be seen. The participant first opened the AuthoritiesConstants class within the com.banking.app.security package before opening the SecurityUtils class within the same package at 87268ms. The first method visible to the user was isAuthenticated within the SecurityUtilsClass followed by a transition to the isCurrentUserInRole method. As indicated by the font size, the second method was visible to the subject for a longer time. Afterwards, the participant navigated to the com.payment.app.security.SecurityUtils class, located within the payment-app Micro-Service at 93978ms. This transition is indicating that the developer could be a novice as the prior opened

com.banking.app.security.SecurityUtils class is handling technical security of the application and not checking any security features regarding e-commerce and the user had to find a class doing so (checking payment card validity). Either there is a lack of understanding what the methods of the classes are doing or the user is thinking, that the two classes with the same name and located within a corresponding package structure, are responsible for different things within the two Micro-Services of the same application. Neither of both possibilities would apply to an experienced programmer. Further indicating that the ana-

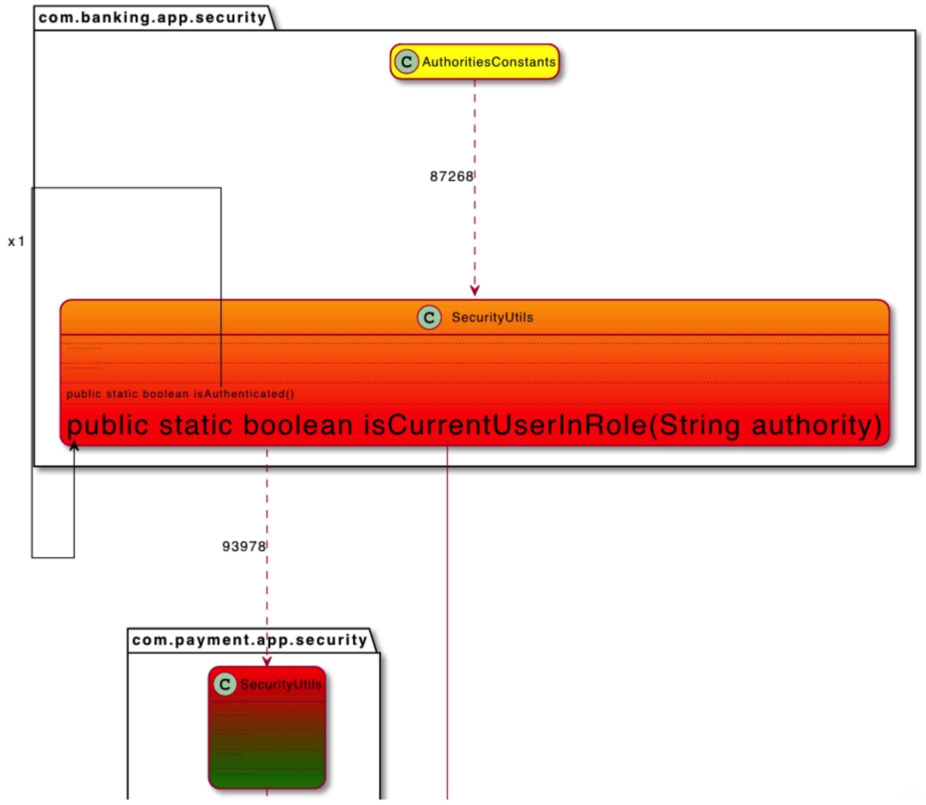


Figure 11: Transitions between first three visible classes spanning 2 packages

lyzed subject is lacking of experience, there are too many transitions between the methods of the PaymentApp class (e.g. 2x main -> logApplicationStartup), which is containing the main method of the payment-app Micro-Service, thus clearly should not be the class the user is looking for. But the methods do not appear to have drawn attention to the user for a long time, as the font-size of the three PaymentApp class methods is relatively small, compared to the long time frame the user kept this class open. The transition to PaymentApp happened at 117004ms, leaving the class happened at 206127ms, meaning the user stayed around 90s within the PaymentApp class before opening another one. Without further knowledge, it could be, that the subject was navigating through the project structure or using the search functionality of the IDE while no method of the PaymentApp class was being looked upon (cf. Figure 12). This state-

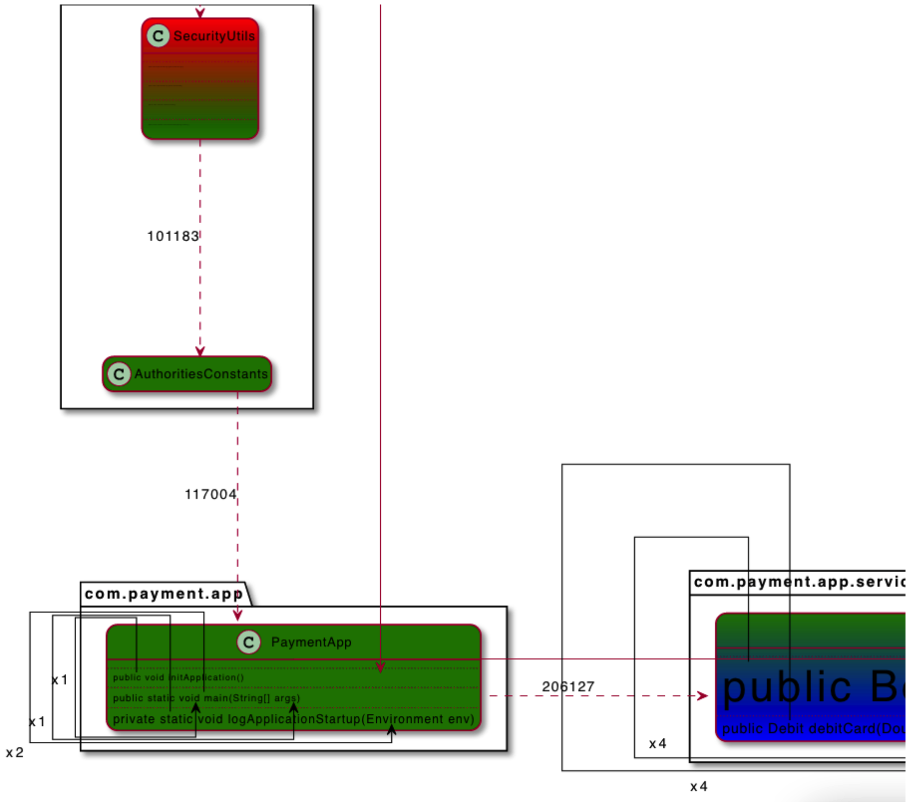

SecurityUtils
101183
AuthoritiesConstants
117004
com.payment.app
PaymentApp
public void initApplication()
public static void main(String[] args)
private static void logApplicationStartup(Environment env)
x1
x1
x2
206127
com.payment.app.servic
public B
public Debit debitCard(Do
x4
x4

Figure 12: Multiple method transitions within PaymentApp class

ment is further acknowledged when looking at (cf. Figure 13). Transitioning from PaymentApp to the CardReaderManager class at 206127ms and leaving it again at 221636ms again, the user looked at validateCard for a long time and scrolled a lot within this class, indicated by the 8 method transitions in total. As a final result, the user decided to solve the task by given the validateCard method name and the CardReaderManager class as an answer to the analyzed task, which class is responsible for validating payment cards.

4.8.2 Eye Fixations Path Diagram

Structure

The eye fixation path diagram is containing all fixations of the subject when looked upon the pre-defined code surface. This data is taken from the Pupil player export and visualized with animated transitions, using D3.js. The trans-

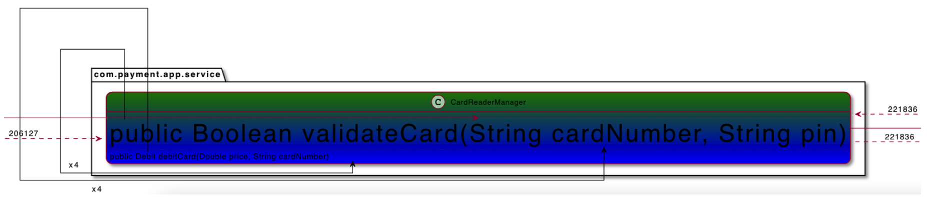

com.payment.app.service
CardReaderManager
public Boolean validateCard(String cardNumber, String pin)
public Debit debitCard(Double price, String cardNumber)
206127
221836
221836
x4
x4

Figure 13: Multiple method transitions within PaymentApp class

itions are animated in real time and are showing the reading patterns of the user when looking at the source code.

Discussion
The result of visualizing the path of eye gaze fixations is leading to the same conclusion as the analysis of the PlantUML diagram, as the visualized pattern shows how the participant is reading the source code almost entirely like reading a book, not skipping to more important parts but strictly reading line per line (cf. Figure 14). More experienced programmers do not follow a linear order when reading source code but follow the executions of source code as well as skipping non-relevant parts by experience [23].

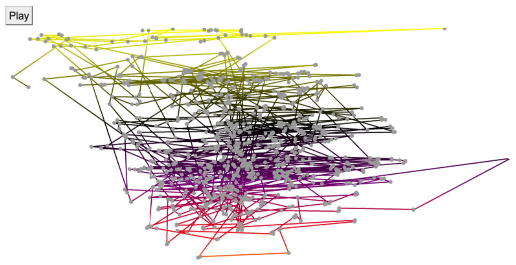


Figure 14: Eye Fixation Path of Participant

4.8.3 Eye Gaze Distribution Diagram

Structure
Similar to the eye fixations path diagram, the input data is taken from the exported Pupil player data. Taken the total video duration as base, the distribution of the eye gazes across the pre-defined surfaces (code editor, navigation, chat, etc.) is visualized using a simple pie chart.

Discussion
The eye gaze distribution analysis revealed that the participant was fixated about a 1/3 of the total task duration within the navigation, thus looking for

relevant classes and finding a way through the multiple Micro-Services (cf. Figure 15).

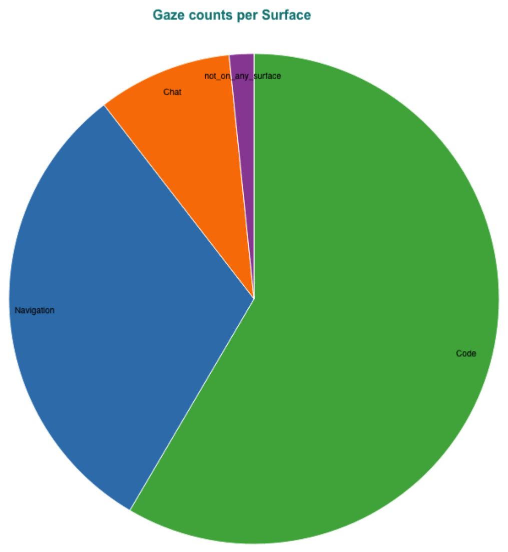


Figure 15: Eye Gaze Distribution of Participant

5 Discussion

This thesis has shown, that it is possible to automate the extraction of source code from videos, to identify matches from the extracted lines to be mapped to methods and classes from the original source code and to visualize the navigation and eye-gaze movements of the user through methods and classes across the duration of the video. Due to the COVID-19 pandemic, neither coming to the university nor discussing the project in person with my supervisor was possible, which affected part of the work on this project. Creating own data to work with, using a similar setting like within the experiment was planned. This would have allowed to set up an ideal environment for getting optimal results.

5.1 Conclusions

The original experiment took place before this thesis was written, therefore no thoughts were made to standardize the environment. One of the major problems with the used input for the application was, that the participants could resize all windows at any time, making it hard to find the correct dimensions for extracting the frames from the screen capture. In some cases, the data could not be used as the size of the code editor was extremely small, thus making it impossible to generate useful output for further processing. The second problem was the usage of a dark IDE theme, although inverting the colors of the video resulted in acceptable results. But using a light theme with black fonts on a white background would have eliminated the need for inverting colors, possibly leading to a better result of the OCR process. Furthermore, the screen capture videos had to be edited before frame extraction, to only contain the task which was analyzed within this thesis. The result of cutting the screen capture must be done carefully to achieve an acceptable level of synchronization with the world video data. This was done using Screenium 3, a non open-source software, primarily used for screen captures but also coming with video editing functionality, allowing to synchronize the two video streams with a difference less than 50ms. Creating a world video and a screen capture for each task with a defined start and end timestamp would have eliminated the need of manually editing the screen capture videos. Another problem was that the needed surfaces to work with (code, navigation) had to be defined prior exporting the needed data for the application and due to the non-standardized environment, each world video had surfaces with differing sizes and had to be handled differently when visualizing the data. To sum up, the following conclusions can be derived from the occurred problems if the presented application should be used to analyze further eye-gaze movement experiments:

- Usage of a light theme, preferably with black fonts on white background
- Fixed sizes of navigation, code and chat window
- Predefined surfaces for each window

- Separate world and screen captures for each task, with defined start/end timestamp
- Separate captures of editor and other trace data for each task, with synced timestamp
- Providing a documentation of the contained data on how to read and use it

I think that the implemented solution should be considered for further experiments as when used with standardized input data, not needing to be prepared extensively per hand, the videoOCR tool can provide an efficient solution to gain better insights into the cognitive processes of developers when working with source code and further development of the videoOCR tool should be encouraged.

5.2 Future Work

Future work could make use of the possibility to train the Tesseract OCR engine with Java source code samples, to achieve better OCR results. The application could be optimized for further eye-tracking experiments if the experiments are done within a standardized environment (e.g. fixed windows sizes, light IDE theme etc.). To open up videoOCR for more use cases, the application of deep learning technologies as shown in [10] could be an option, eliminating the constraints of manually selecting the bounding boxes of the output frames.

References

[1] Evaluate ocr output quality with character error rate (cer) and word error rate (wer). https://towardsdatascience.com/evaluating-ocr-output-quality-with-character-error-rate-cer-and-word-error-rate-wer-853175297510. Accessed: 2021-08-02.

[2] Hebrew ocr (optical character recognition) library. https://github.com/yaacov/hebocr. Accessed: 2021-03-20.

[3] Links to awesome ocr projects. https://github.com/kba/awesome-ocr. Accessed: 2021-03-20.

[4] microservice-sales-app github repository. https://github.com/debbienuta/microservice-sales-app. Accessed: 2021-09-08.

[5] Overview of text similarity metrics in python. https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50. Accessed: 2021-09-08.

[6] sales-app github repository. https://github.com/debbienuta/sales-app. Accessed: 2021-09-08.

[7] Stats - central repository for maven, gradle, kotlin, scala and more. https://search.maven.org/stats. Accessed: 2021-03-14.

[8] Texterkennung. https://de.wikipedia.org/wiki/Texterkennung. Accessed: 2021-03-20.

[9] Traineddata files for version 4.00 +. https://tesseract-ocr.github.io/tessdoc/Data-Files.html. Accessed: 2021-08-03.

[10] Alahmadi, M., Hassel, J., Parajuli, B., Haiduc, S., and Kumar, P. Accurately predicting the location of code fragments in programming video tutorials using deep learning. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering* (2018), pp. 2–11.

[11] Balalaie, A., Heydarnoori, A., and Jamshidi, P. Migrating to cloud-native architectures using microservices: an experience report. In *European Conference on Service-Oriented and Cloud Computing* (2015), Springer, pp. 201–215.

[12] Buxton, J. N., and Randell, B. *Software engineering techniques: report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969.* NATO Science Committee, 1970.

[13] Carrasco, R. C. An open-source ocr evaluation tool. In *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage* (New York, NY, USA, 2014), DATeCH '14, Association for Computing Machinery, p. 179–184.

[14] Cohen, W., Ravikumar, P., and Fienberg, S. A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation* (2003), vol. 3, pp. 73–78.

[15] Coplien, J. O. *Advanced C++ programming styles and idioms.* Addison-Wesley Longman Publishing Co., Inc., 1991.

[16] Gamma, E., Helm, R., Johnson, R., Vlissides, J., and Patterns, D. *Elements of reusable object-oriented software*, vol. 99. Addison-Wesley Reading, Massachusetts, 1995.

[17] Kircher, M., Jain, P., et al. Lookup.

[18] Kruchten, P., Obbink, H., and Stafford, J. The past, present, and future for software architecture. *Software, IEEE 23* (04 2006), 22 – 30.

[19] Kruchten, P. B. The 4+ 1 view model of architecture. *IEEE software 12*, 6 (1995), 42–50.

[20] MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R., and Hamilton, B. A. Reference model for service oriented architecture 1.0. *OASIS standard 12*, S 18 (2006).

[21] Newman, S. *Building microservices.* " O'Reilly Media, Inc.", 2021.

[22] Niwattanakul, S., Singthongchai, J., Naenudorn, E., and Wanapu, S. Using of jaccard coefficient for keywords similarity. In *Proceedings of the international multiconference of engineers and computer scientists* (2013), vol. 1, pp. 380–384.

[23] Peitek, N., Siegmund, J., and Apel, S. What drives the reading order of programmers? an eye tracking study. In *Proceedings of the 28th International Conference on Program Comprehension* (2020), pp. 342–353.

[24] Perry, D. E., and Wolf, A. L. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes 17*, 4 (1992), 40–52.

[25] Ponzanelli, L., Bavota, G., Mocci, A., Di Penta, M., Oliveto, R., Russo, B., Haiduc, S., and Lanza, M. Codetube: extracting relevant fragments from software development video tutorials. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)* (2016), IEEE, pp. 645–648.

[26] Schmidt, D. C., Stal, M., Rohnert, H., and Buschmann, F. *Pattern-oriented software architecture, patterns for concurrent and networked objects*, vol. 2. John Wiley & Sons, 2013.

[27] Shadija, D., Rezai, M., and Hill, R. Towards an understanding of microservices. In *2017 23rd International Conference on Automation and Computing (ICAC)* (2017), IEEE, pp. 1–6.

[28] Simhandl, G., Paulweber, P., and Zdun, U. Design of an executable specification language using eye tracking. In *6th International Workshop on Eye Movements in Programming (EMIP)* (May 2019).

[29] Stal, M. Using architectural patterns and blueprints for service-oriented architecture. *IEEE software 23*, 2 (2006), 54–61.

[30] Zimmermann, O. Microservices tenets. *Computer Science-Research and Development 32*, 3 (2017), 301–310.

A Deployment and Usage of videoOCR

The following steps must be taken to deploy the VideoOCR application to a docker container. Prerequisites: Java 11, Maven and Docker must be installed prior.

1. Clone the repository
2. Open a terminal and change to the project's root directory
3. Run "mvn clean install" to execute the clean and install Maven goals
4. Run "docker build -t harb/ocr ." to build the docker image (this can take a while, use "--progress plain" as parameter for verbose progress output)
5. Run "docker run –name ocr -t -i -p 8083:8083 harb/ocr" to run the container
6. Restart the container by running "docker stop ocr" and then running "docker start ocr -a" (not restarting the container bugs the download function, no other solution to this problem could be found)

After deployment, the application is running in a docker container with port 8083 exposed. Prerequisites for visualizing data: The editor trace file must be prepared manually to only contain the files of the analyzed task and the timestamps must be normalized to match the timestamps of the video. Furthermore the gaze distribution csv file must be normalized. The listing below is showing the normalized csv.

Listing 9: normal_surface_gaze_distribution.csv

```
surface_name,gaze_count,total_count
Code,6731,11641
Navigation,3577,11641
Chat,1020,11641
not_on_any_surface,187,11641
```

The default usage is described by the following steps.

1. Open a web browser and navigate to "http://localhost:8083/video".
2. Click on "Durchsuchen" and chose a video file to upload (filename for micro-service videos must start with "micro")
3. Click on "Upload" and wait for the upload to be finished
4. Beneath the "Extract Frames" button, enter the width and height of the frames to be extracted. Also, enter the starting coordinates on the x and y axis (calculated from the upper left corner)
5. Click on "Extract Frames" to extract the frames and wait for the process to finish

6. Click on "Perform OCR" to extract the text from the frames and wait for the process to finish
7. Besides the "Upload Pupil Files" button, select the editor trace file and upload it (filename must be "trace_editor.txt").
8. Besides the "Upload Pupil Files" button, select the gaze distributions csv file and upload it (filename must be "normal_surface_gaze_distribution.csv").
9. Besides the "Upload Pupil Files" button, select the gaze distributions csv file and upload it (filename must be "fixations_on_surface_Code.csv").
10. Click "Create Viz Data" and wait for the process to finish.
11. The generated files can be downloaded by clicking onto them (e.g the plantuml.txt and plantuml.svg files are just generated and not further displayed)
12. Navigating to "Path Visualization", choosing the video and clicking "Load", visualizes the eye-gaze fixations on the code surface
13. Navigating to "Eye Gaze Duration", choosing the video and clicking "Load", visualizes the eye-gaze distributions across the surfaces based on the total video duration.

B Screenshots and Diagrams of videoOCR

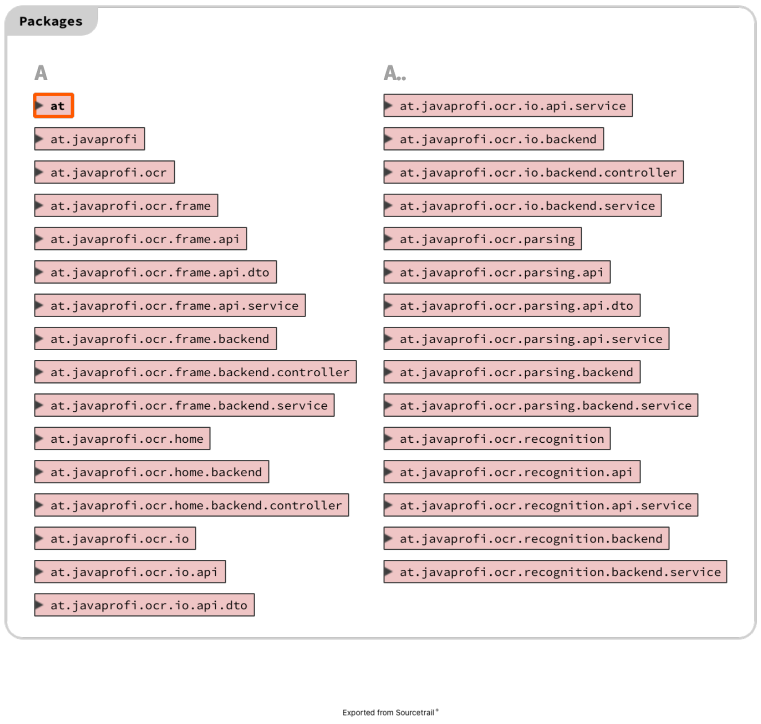

Packages
A
at
at.javaprofi
at.javaprofi.ocr
at.javaprofi.ocr.frame
at.javaprofi.ocr.frame.api
at.javaprofi.ocr.frame.api.dto
at.javaprofi.ocr.frame.api.service
at.javaprofi.ocr.frame.backend
at.javaprofi.ocr.frame.backend.controller
at.javaprofi.ocr.frame.backend.service
at.javaprofi.ocr.home
at.javaprofi.ocr.home.backend
at.javaprofi.ocr.home.backend.controller
at.javaprofi.ocr.io
at.javaprofi.ocr.io.api
at.javaprofi.ocr.io.api.dto
A..
at.javaprofi.ocr.io.api.service
at.javaprofi.ocr.io.backend
at.javaprofi.ocr.io.backend.controller
at.javaprofi.ocr.io.backend.service
at.javaprofi.ocr.parsing
at.javaprofi.ocr.parsing.api
at.javaprofi.ocr.parsing.api.dto
at.javaprofi.ocr.parsing.api.service
at.javaprofi.ocr.parsing.backend
at.javaprofi.ocr.parsing.backend.service
at.javaprofi.ocr.recognition
at.javaprofi.ocr.recognition.api
at.javaprofi.ocr.recognition.api.service
at.javaprofi.ocr.recognition.backend
at.javaprofi.ocr.recognition.backend.service
Exported from Sourcetrail®

Figure 16: Package structure of videoOCR

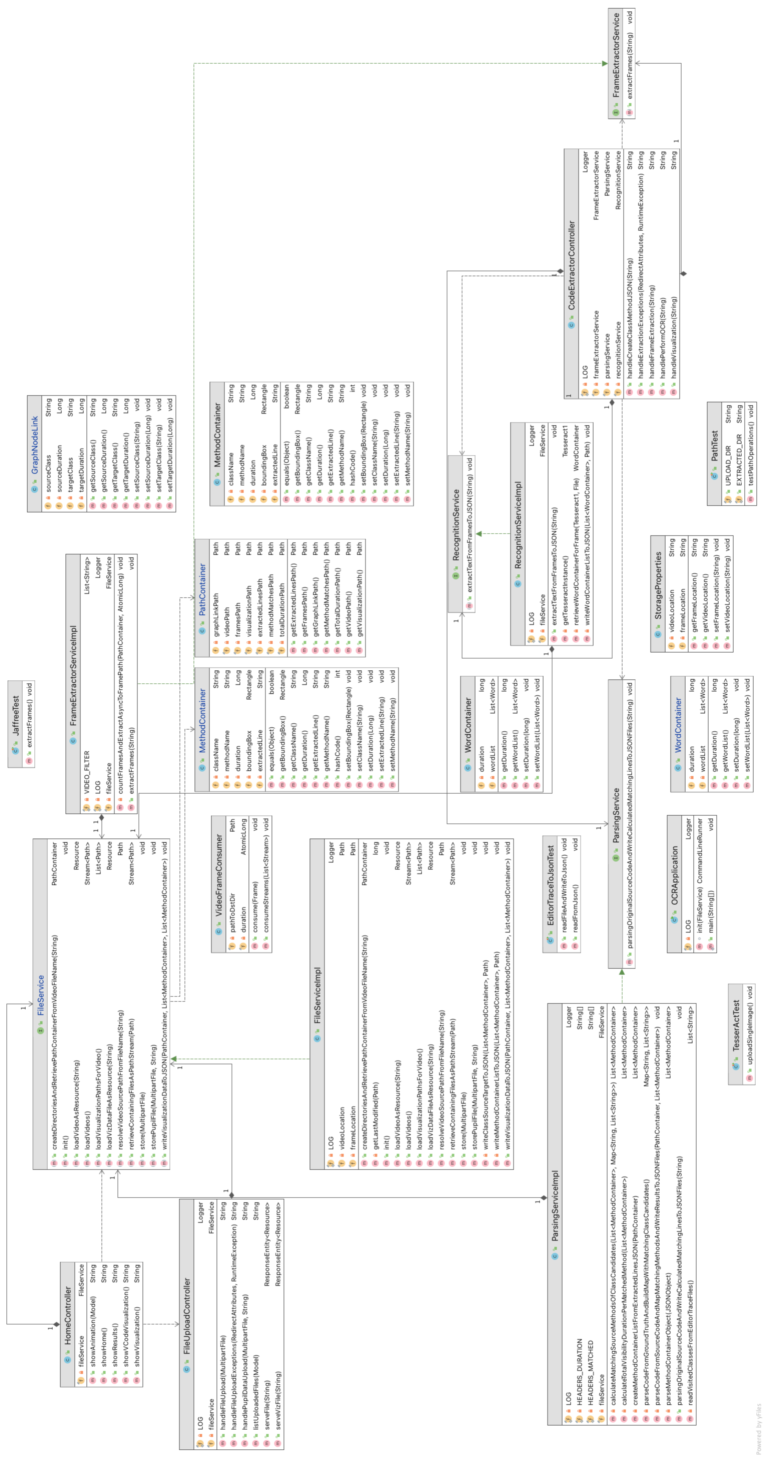


Figure 17: UML class diagram

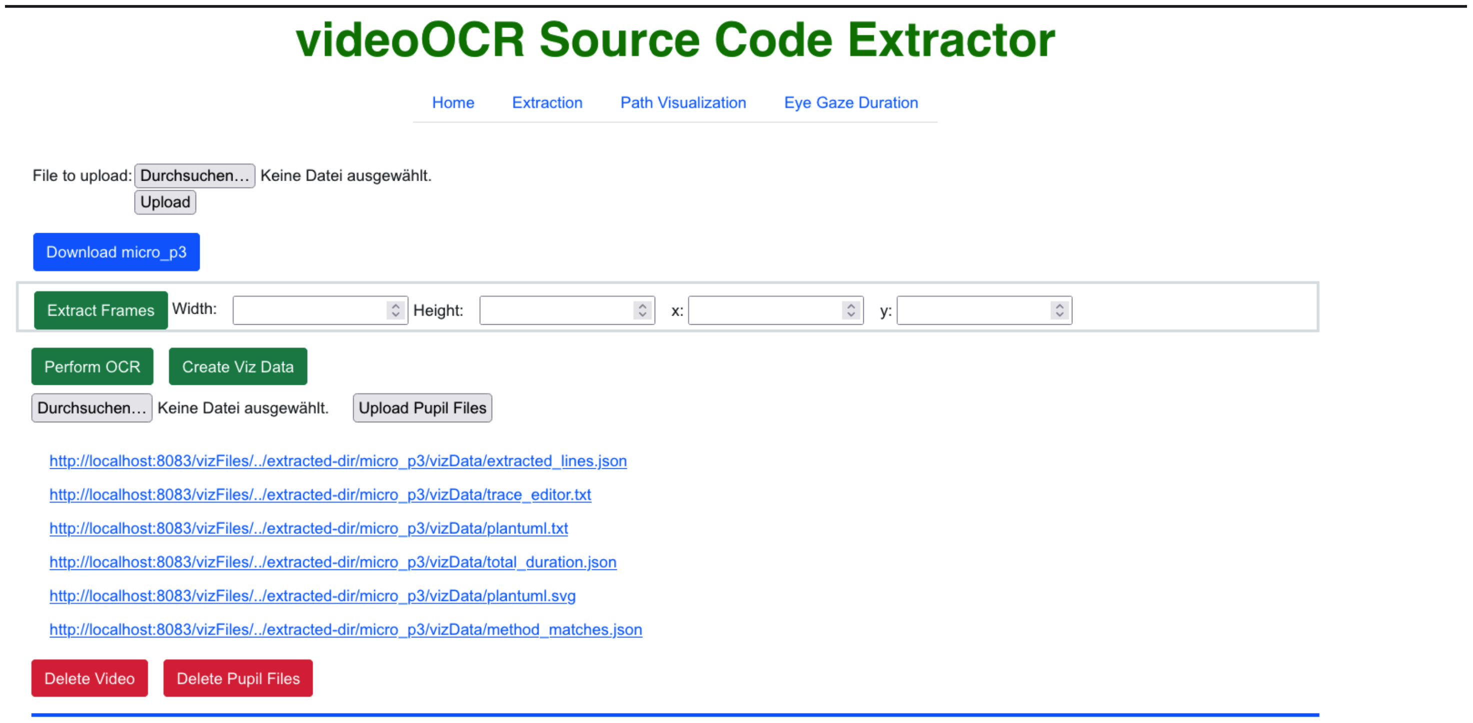

videoOCR Source Code Extractor
Home
Extraction
Path Visualization
Eye Gaze Duration
File to upload:
Durchsuchen…
Keine Datei ausgewählt.
Upload
Download micro_p3
Extract Frames
Width:
Height:
x:
y:
Perform OCR
Create Viz Data
Durchsuchen…
Keine Datei ausgewählt.
Upload Pupil Files
http://localhost:8083/vizFiles/../extracted-dir/micro_p3/vizData/extracted_lines.json
http://localhost:8083/vizFiles/../extracted-dir/micro_p3/vizData/trace_editor.txt
http://localhost:8083/vizFiles/../extracted-dir/micro_p3/vizData/plantuml.txt
http://localhost:8083/vizFiles/../extracted-dir/micro_p3/vizData/total_duration.json
http://localhost:8083/vizFiles/../extracted-dir/micro_p3/vizData/plantuml.svg
http://localhost:8083/vizFiles/../extracted-dir/micro_p3/vizData/method_matches.json
Delete Video
Delete Pupil Files

Figure 18: Upload and Main Page

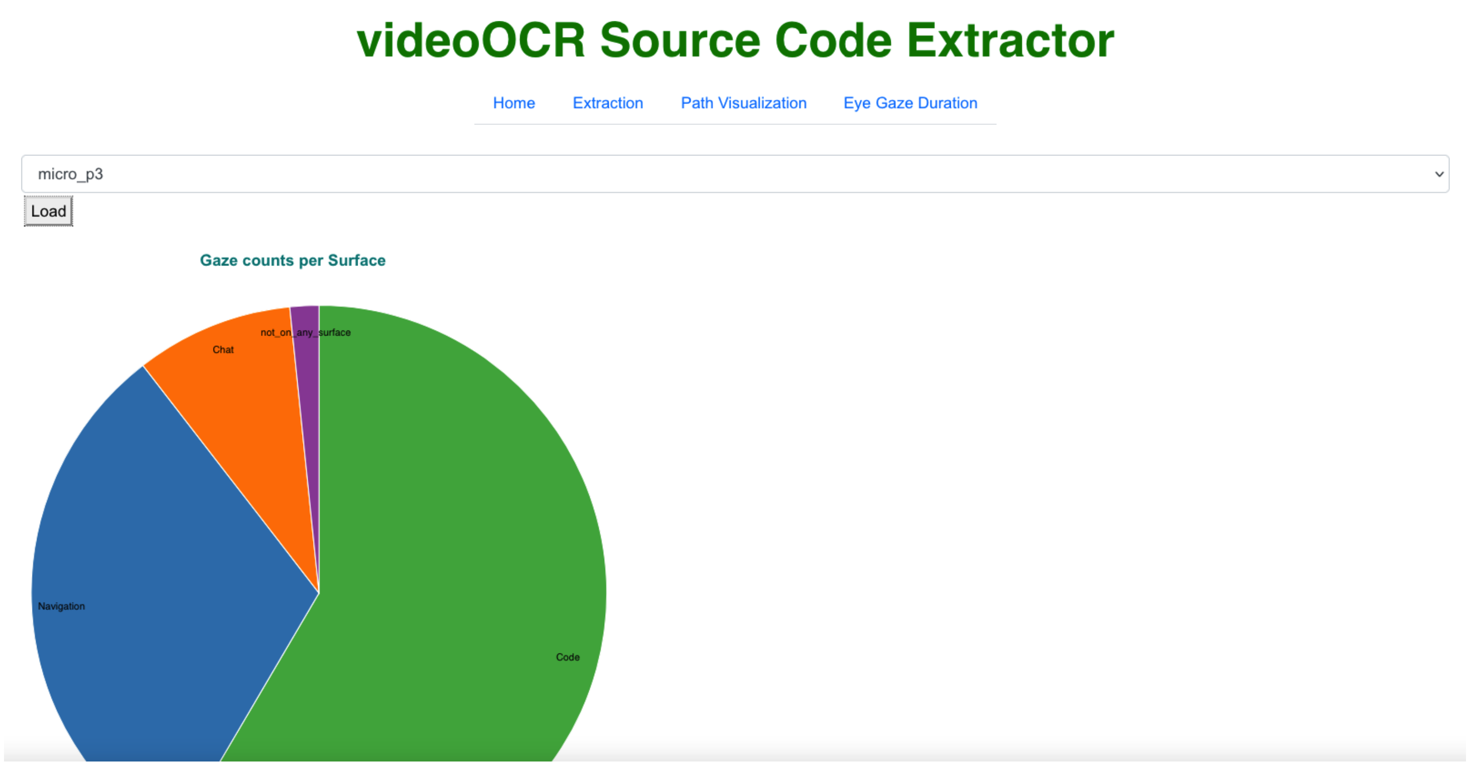


Figure 19: Gaze Surface Distribution Page

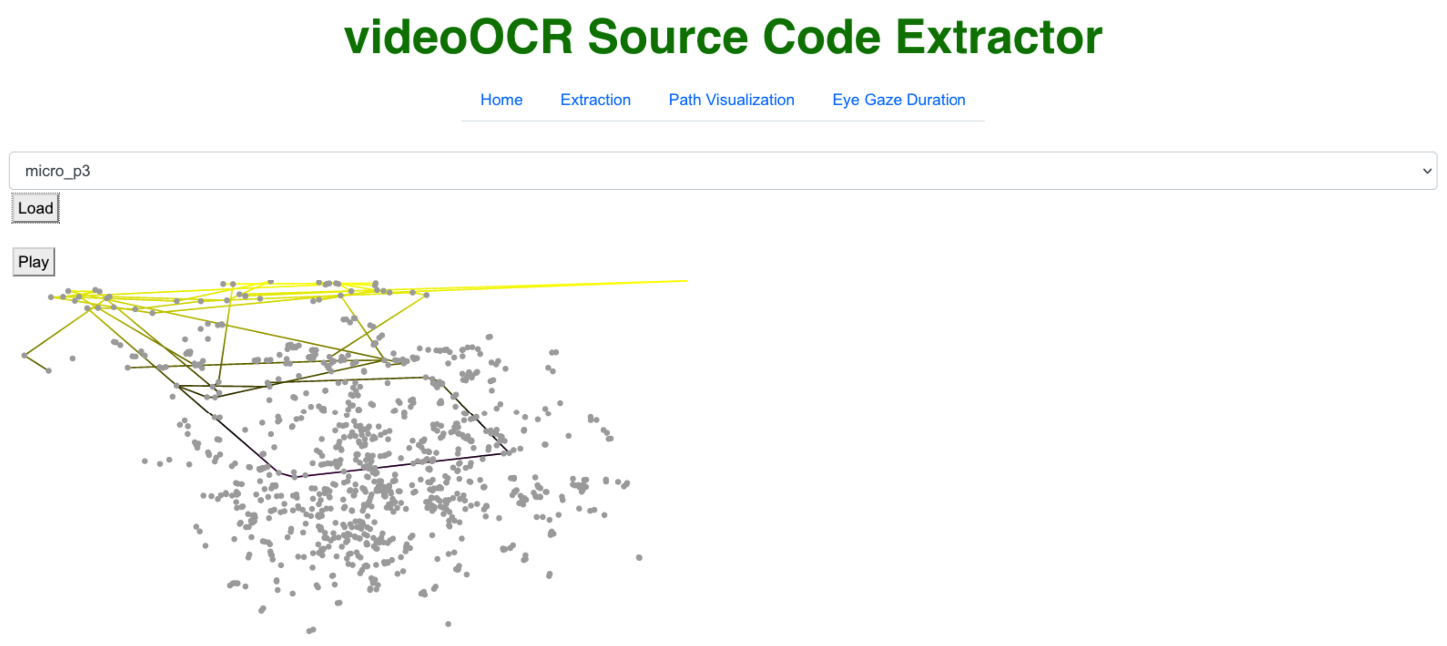


Figure 20: Path Visualization Page

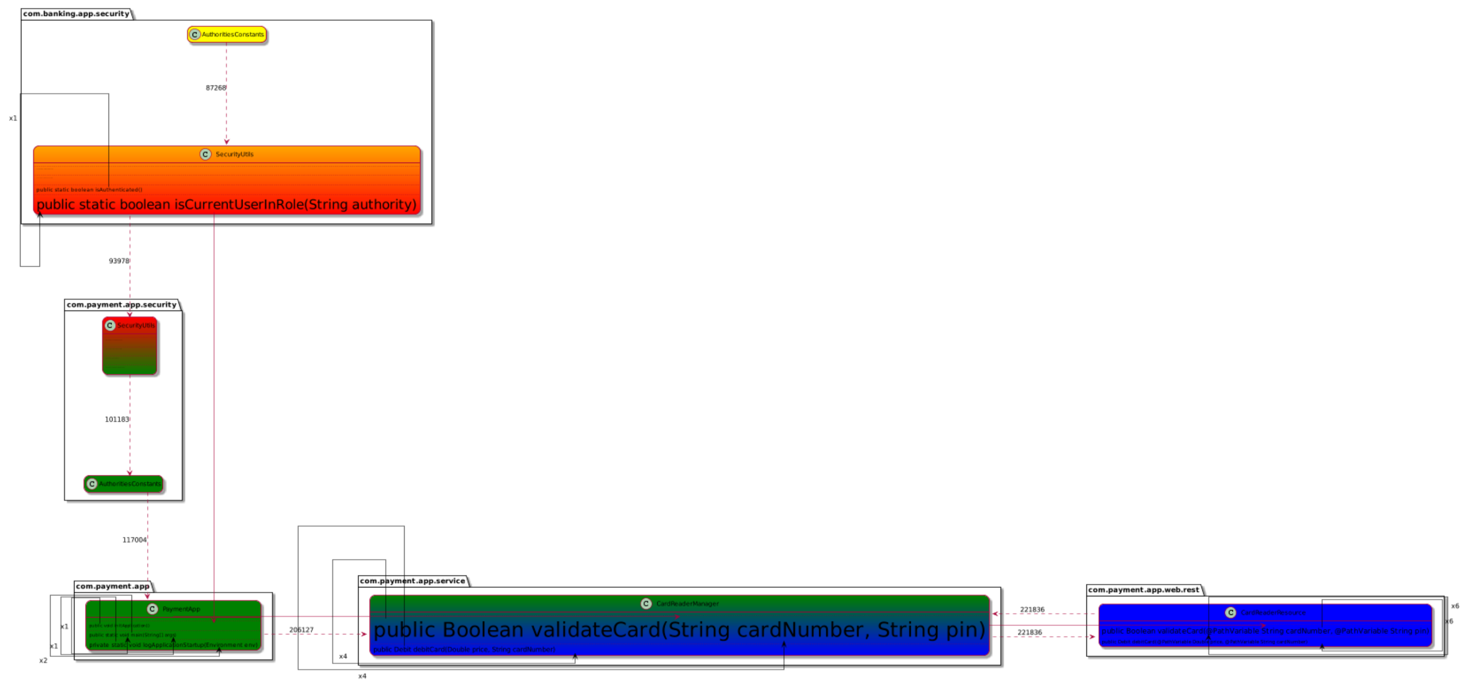
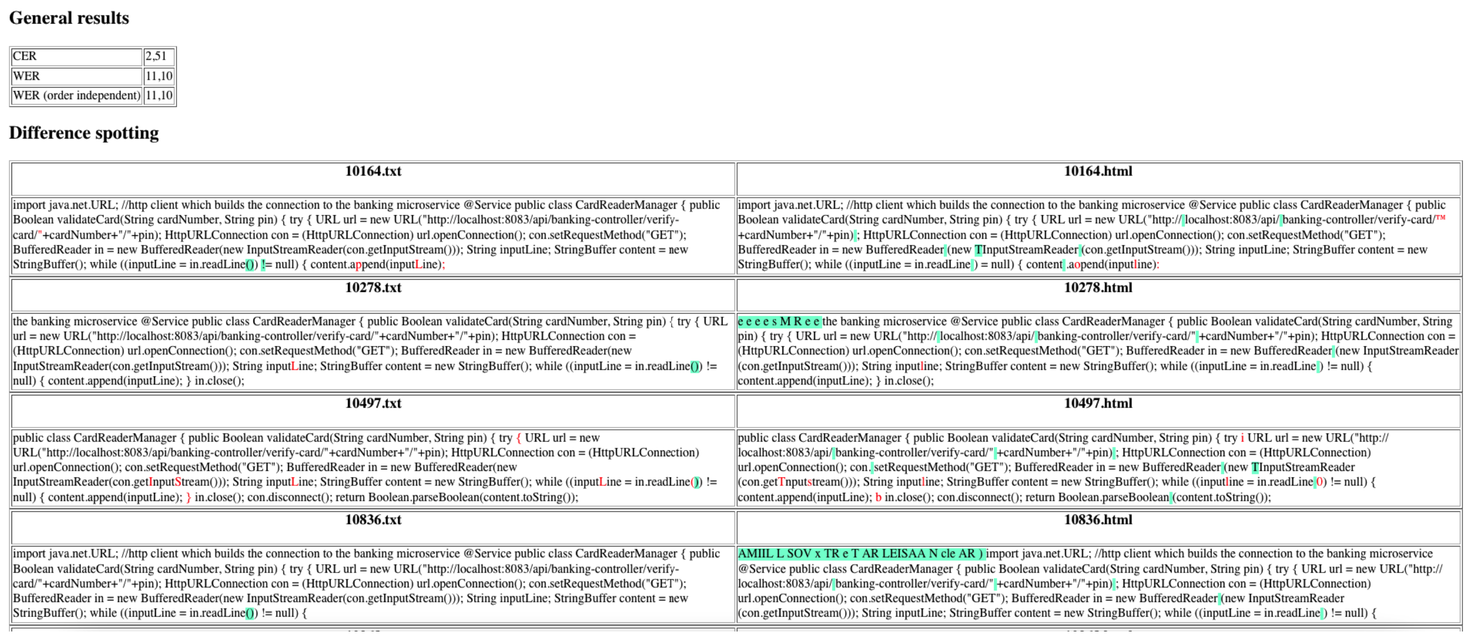


Figure 21: Scaled PlantUML Diagram



General results

CER	2,51
WER	11,10
WER (order independent)	11,10

Difference spotting

10164.txt	10164.html
import java.net.URL; //http client which builds the connection to the banking microservice @Service public class CardReaderManager { public Boolean validateCard(String cardNumber, String pin) { try { URL url = new URL("http://localhost:8083/api/banking-controller/verify-card/"+cardNumber+"/"+pin); HttpURLConnection con = (HttpURLConnection) url.openConnection(); con.setRequestMethod("GET"); BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream())); String inputLine; StringBuffer content = new StringBuffer(); while ((inputLine = in.readLine()) != null) { content.append(inputLine);	import java.net.URL; //http client which builds the connection to the banking microservice @Service public class CardReaderManager { public Boolean validateCard(String cardNumber, String pin) { try { URL url = new URL("http:// localhost:8083/api/ banking-controller/verify-card/™ +cardNumber+"/"+pin) ; HttpURLConnection con = (HttpURLConnection) url.openConnection(); con.setRequestMethod("GET"); BufferedReader in = new BufferedReader (new TInputStreamReader (con.getInputStream())); String inputLine; StringBuffer content = new StringBuffer(); while ((inputLine = in.readLine) = null) { content .aopend(inputline):

10278.txt	10278.html
the banking microservice @Service public class CardReaderManager { public Boolean validateCard(String cardNumber, String pin) { try { URL url = new URL("http://localhost:8083/api/banking-controller/verify-card/"+cardNumber+"/"+pin); HttpURLConnection con = (HttpURLConnection) url.openConnection(); con.setRequestMethod("GET"); BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream())); String inputLine; StringBuffer content = new StringBuffer(); while ((inputLine = in.readLine()) != null) { content.append(inputLine); } in.close();	e e e e s M R e e the banking microservice @Service public class CardReaderManager { public Boolean validateCard(String cardNumber, String pin) { try { URL url = new URL("http:// localhost:8083/api/ banking-controller/verify-card/" +cardNumber+"/"+pin); HttpURLConnection con = (HttpURLConnection) url.openConnection(); con.setRequestMethod("GET"); BufferedReader in = new BufferedReader (new InputStreamReader (con.getInputStream())); String inputline; StringBuffer content = new StringBuffer(); while ((inputLine = in.readLine) != null) { content.append(inputLine); } in.close();

10497.txt	10497.html
public class CardReaderManager { public Boolean validateCard(String cardNumber, String pin) { try { URL url = new URL("http://localhost:8083/api/banking-controller/verify-card/"+cardNumber+"/"+pin); HttpURLConnection con = (HttpURLConnection) url.openConnection(); con.setRequestMethod("GET"); BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream())); String inputLine; StringBuffer content = new StringBuffer(); while ((inputLine = in.readLine()) != null) { content.append(inputLine); } in.close(); con.disconnect(); return Boolean.parseBoolean(content.toString());	public class CardReaderManager { public Boolean validateCard(String cardNumber, String pin) { try i URL url = new URL("http:// localhost:8083/api/ banking-controller/verify-card/" +cardNumber+"/"+pin) ; HttpURLConnection con = (HttpURLConnection) url.openConnection(); con. setRequestMethod("GET"); BufferedReader in = new BufferedReader (new TInputStreamReader (con.getTnputstream())); String inputline; StringBuffer content = new StringBuffer(); while ((inputline = in.readLine 0) != null) { content.append(inputLine); b in.close(); con.disconnect(); return Boolean.parseBoolean (content.toString());

10836.txt	10836.html
import java.net.URL; //http client which builds the connection to the banking microservice @Service public class CardReaderManager { public Boolean validateCard(String cardNumber, String pin) { try { URL url = new URL("http://localhost:8083/api/banking-controller/verify-card/"+cardNumber+"/"+pin); HttpURLConnection con = (HttpURLConnection) url.openConnection(); con.setRequestMethod("GET"); BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream())); String inputLine; StringBuffer content = new StringBuffer(); while ((inputLine = in.readLine()) != null) {	AMIIL L SOV x TR e T AR LEISAA N cle AR) import java.net.URL; //http client which builds the connection to the banking microservice @Service public class CardReaderManager { public Boolean validateCard(String cardNumber, String pin) { try { URL url = new URL("http:// localhost:8083/api/ banking-controller/verify-card/" +cardNumber+"/"+pin) ; HttpURLConnection con = (HttpURLConnection) url.openConnection(); con.setRequestMethod("GET"); BufferedReader in = new BufferedReader (new InputStreamReader (con.getInputStream())); String inputLine; StringBuffer content = new StringBuffer(); while ((inputLine = in.readLine) != null) {

Figure 22: Sample Output of OCR Evaluation Tool