Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Weiterentwicklung einer Software-Suite für das Aufnehmen und Abspielen von Testvideos

Bachelor's Thesis

im Studiengang Computer Science

von

Shaochen Wu

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. Jil Ann-Christin Klünder
Betreuer: M. Sc. Jianwei Shi

Hannover, 15.09.2023

Declaration of Authorship

I hereby declare that the Bachelor's Thesis is my unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the thesis in digital form can be examined for the use of unauthorized aid and determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources, I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published.

Hannover, den 15.09.2023

Shaochen Wu

Zusammenfassung

In großen Softwareprojekten und globalen Entwicklungen ist eine häufige Herausforderung die unzureichende Kommunikation zwischen den Stakeholdern. Aufgrund des Arbeitsablaufs und der Herangehensweise in der agilen Entwicklung können nur bestimmte Stakeholder Feedback geben. Dies kann zu Missverständnissen bezüglich der Projektanforderungen führen und potenziell dazu führen, dass Softwareprodukte geliefert werden, die nicht den Erwartungen der Stakeholder oder Kunden entsprechen. Um dieses Problem zu lösen, ist die Erstellung und Verwendung von Videos zur Einholung von Feedback von Stakeholdern in Kombination mit dem BDD-Framework eine geeignete Lösung. Die dafür verwendeten Tools sind ScreenTracer und ScreenTracerViewer. Es gibt jedoch immer noch viele Probleme und Verbesserungsmöglichkeiten bei ScreenTracer und ScreenTracerViewer. Daher ist die kontinuierliche Weiterentwicklung von ScreenTracer und ScreenTracerViewer geplant. Der aktuelle Treiber für ScreenTracer weist Instabilitäten auf und soll durch eine robustere Alternative ersetzt werden. Darüber hinaus ermöglichen Verbesserungen an ScreenTracer die Generierung sowohl einer neuen Video-Datei im neuen Format als auch einer WebVTT-Datei mit Ereignisinformationen, die mit dem Video synchronisiert sind. Die Ereignisinformationen werden dem Gherkin-Format entsprechen und Zeitstempel enthalten, wann die Ereignisse im Video auftreten. Nach den Verbesserungen an ScreenTracer verschiebt sich der Fokus dieses Projekts auf die Verbesserung von ScreenTracerViewer. Derzeit stehen zwei Versionen von Screentracerviewer für weitere Entwicklung zur Verfügung: die Originalversion und eine HTML-Version. Die endgültige Wahl für dieses Projekt ist die HTML-Version. Diese Entscheidung basiert auf der Prämisse, die gleiche Funktionalität zu erreichen. Unter dieser Prämisse ist die Entwicklung der HTML-Version von Screentracerviewer einfacher und besser geeignet, sie möglicherweise in Zukunft zu einer Browser-Erweiterung umzuwandeln. Derzeit bietet die HTML-Version von ScreenTracerViewer eingeschränkte Funktionalität und verwendet ausschließlich ein Frontend-Framework eines Drittanbieters, um eine Benutzeroberfläche bereitzustellen. Es sind umfangreiche Erweiterungen geplant, um Benutzern zu ermöglichen, ausführbare Dateien mit ScreenTracer zu öffnen, automatisch Video- und WebVTT-Dateien zu generieren. Anschließend können Benutzer Screen-

TracerViewer verwenden, um mehrere lokale Video-Dateien zu öffnen und gleichzeitig die entsprechenden Gherkin-Spezifikationen innerhalb der Benutzeroberfläche von ScreenTracerViewer anzuzeigen. Dieser umfassende Ansatz gewährleistet, dass alle Teammitglieder, Kunden und Stakeholder in Echtzeit über den Fortschritt der Softwareentwicklung und deren Einhaltung aller Softwareanforderungen informiert bleiben.

Abstract

In large software projects and global development, a common challenge is inadequate communication among stakeholders. Due to the workflow and approach of agile development, only a subset of stakeholders can provide feedback. This can lead to misunderstandings of project requirements, potentially resulting in the delivery of software products that do not meet stakeholder or customer expectations. For this issue, creating and using videos to obtain feedback from stakeholders, combined with the BDD framework, is a viable solution, and the tools used for creating and using videos are ScreenTracer and ScreenTracerViewer. However, there are still many issues and areas for improvement with ScreenTracer and ScreenTracerViewer. Therefore, ongoing development of ScreenTracer and ScreenTracerViewer is planned. The current driver for ScreenTracer exhibits instability and is set to be replaced by a more robust alternative. Additionally, improvements to ScreenTracer will enable it to generate both a new format video file and a WebVTT file containing event information synchronized with the video. The event information will adhere to the Gherkin format and include timestamps of when the events occur in the video. Following the enhancements to ScreenTracer, the focus of this project will shift to the improvement of ScreenTracerViewer. There are currently two versions of Screentracerviewer available for further development: the original version and an HTML version. The final choice for this project is the HTML version. This decision is based on the premise of achieving the same functionality, under this premise, developing the HTML version of Screentracerviewer is simpler and more conducive to potentially turning it into a browser extension in the future. Presently, ScreenTracerViewer of HTML version offers limited functionality, employing a third-party frontend framework solely for providing a user interface. Extensive feature additions are planned to empower users to open executable files with ScreenTracer, automatically generating video and WebVTT files. Subsequently, users can utilize ScreenTracerViewer to open multiple local video files and concurrently view corresponding Gherkin specifications within the ScreenTracerViewer interface. This comprehensive approach ensures that all team members, customers, and stakeholders can stay updated in real-time on the software development progress and its compliance with all software requirements.

Contents

1 Introduction **1**

1.1 Research Aims and Objectives 1

1.2 Structure of the thesis 2

2 Fundamentals **3**

2.1 Related Works 3

2.1.1 Challenges of project management in global software development: A client-vendor analysis 3

2.1.2 State of Practice of User-Developer Communication in Large-Scale IT Projects 3

2.1.3 BDD in Action: Behavior-driven development for the whole software lifecycle 4

2.1.4 Using GUI Test Videos to Obtain Stakeholders' Feedback 4

2.1.5 Who tested my software? Testing as an organizationally cross-cutting activity 4

2.1.6 Stakeholder participation in the development of an electronic medical record system in Malawi 4

2.2 BDD: Behavior-Driven Development 5

2.3 SharpAvi 5

2.4 SpecFlow 5

2.5 LivingDoc 6

2.6 Selenium IDE 7

2.7 ScreenTracer 7

3 Requirements **9**

3.1 Design process 9

3.2 Stakeholders 9

3.3 Mandatory Requirements 10

3.4 Optional Requirements 11

4 Implementation **13**

4.1 R1: When the user ends the screen recording, ScreenTracer shall be able to generate a video and automatically store the video in .AVI format. 13
4.2 R2: When the user ends the screen recording, ScreenTracer shall be able to automatically store a link between time periods of the video and the Gherkin specification in a file (e.g. json, csv). 14
4.3 R3: When a video file is playing in ScreenTracerViewer, ScreenTracerViewer shall provide the user with the ability to also see the Gherkin specification of this video (Gherkin specification uses the Given-When-Then grammar to describe what is happening in the video). 17
4.3.1 How to convert scenario files into test codes 18
4.4 R4: When multiple video files are open in ScreenTracerViewer, ScreenTracerViewer shall separate the multiple videos by a thick vertical line in the progress bar and provide the user with the ability to play multiple videos under one view in ScreenTracerViewer. 19
4.5 R5: When multiple video files are open in ScreenTracerViewer, ScreenTracerViewer shall provide the user with the ability to define the playback order of videos. 21

5 Evaluation and Discussion 23
5.1 Test Environment 23
5.2 Test Case 23
5.3 Test Results 34
5.4 The Practical Use of ScreenTracer and ScreenTracerViewer . 35
5.4.1 The benefits of using ScreenTracer and ScreenTracerViewer 36
5.5 Software development process of BDD 36
5.6 Gherkin and Test Case 38
5.7 The comparison between BDD- and traditional software development process 40

6 Conclusion 43
6.1 Conclusion 43
6.2 Future Work 44

A Appendix A 45

Chapter 1

Introduction

A common challenge encountered in large software projects and global software development is inadequate communication among stakeholders. In agile software development, feedback is typically obtained during review meetings, such as sprint planning meetings. However, these meetings often involve only the on-site customer and the development team, excluding other relevant stakeholders from providing feedback. This limitation can result in misunderstandings regarding project requirements, potentially leading to the implementation of software with incorrect or omitted functionality. Consequently, this can lead to dissatisfaction among stakeholders or customers. To effectively tackle this problem, ScreenTracer and ScreenTracerViewer will be developed and improved to create and utilize videos for obtaining feedback from stakeholders.

1.1 Research Aims and Objectives

The existing ScreenTracer has the capability to run executable files and record the screen during their execution. The screen recording process terminates simultaneously with the completion of its execution of the executable file and automatically generates a video file. However, the current driver is choppy. On the other hand, the current state of ScreenTracerViewer is rather limited in terms of functionality. It primarily utilizes a third-party framework to establish a basic user interface. The main goal of this project is to develop and improve ScreenTracer and ScreenTracerViewer. In the initial phase, the primary objective is to enhance Screentracer for improved screen recording stability. In the second phase, the continued development of the existing ScreenTracerViewer aims to enable customers and stakeholders to use it for verifying whether the software complies with all the software requirements. The objective is to enhance Screentracer and Screentracerviewer to better assist software development teams in delivering software products that satisfy customers and stakeholders.

1.2 Structure of the thesis

This thesis contains seven chapters. Chapter 1 introduces readers to the goals and objectives of this thesis and gives readers the structure of this thesis. Chapter 2 lists the related works that this thesis is based on and explains the basic fundamentals. Chapter 3 lists all main and optional requirements that are set for ScreenTracer and ScreenTracerViewer during the development phase. Chapter 4 gives the ideas and processes of implementing all the mandatory requirements. Chapter 5 describes the completion status of this project through Test Cases and collects additional summaries, comparisons, and explanations of models and processes based on searched and read information and related work. Chapter 6 reviews the work and proposes ideas on how future work can be done to continue developing ScreenTracer and ScreenTracerViewer.

Chapter 2

Fundamentals

This Chapter lists and introduces the works related to this thesis. These related works will serve as background material to better explain the key themes of this work.

2.1 Related Works

2.1.1 Challenges of project management in global software development: A client-vendor analysis

Global Software Development (GSD) is the process of developing software by different teams located in various parts of the globe. This paper, authored by Niazi et al. [1], identifies challenges from the perspectives of both clients and vendors that could potentially disrupt the successful management of GSD projects. The challenges identified in this article are also the ones that this project aims to address, as reflected in Chapter 1.

2.1.2 State of Practice of User-Developer Communication in Large-Scale IT Projects

Abelein and Paech [2] conducted a series of semi-structured interviews with twelve experts, and the results indicated that direct communication between users and developers was limited. These interviews helped in understanding the current practices and issues, confirming the need for improved communication in large IT projects. This article has provided me with a better understanding of the role and significance of Screentracerviewer in development.

2.1.3 BDD in Action: Behavior-driven development for the whole software lifecycle

Smart and Molak [3] wrote this book to introduce BDD. Readers can learn BDD step-by-step through this book. The author also shows many useful techniques and tools in this book to help readers use BDD better in actual development process. The content of this book is reflected in both Chapter 6, Section 6.4, titled "Software Development Process of BDD," and the present Chapter, Section 2.2, titled "BDD - Behavior-Driven Development."

2.1.4 Using GUI Test Videos to Obtain Stakeholders' Feedback

In software projects, stakeholders can give valuable feedback on software demonstrations. However, in agile software development, only the on-site customer and the development team attend such meetings, and other stakeholders cannot give feedback. This can lead to misunderstandings regarding the requirements, and then the implemented software can have wrong or missing functionality. This will dissatisfied stakeholders or customers. For this problem, Shi et al. [4] gave a solution, which is to obtain feedback from stakeholders by creating (through existing successful GUI tests) and using videos. This project is a practical implementation of this idea.

2.1.5 Who tested my software? Testing as an organizationally cross-cutting activity

To gain a better understanding of how testing is conducted in real-world environments, Mäntylä et al. [5] conducted interviews with three software companies and completed a research. The research revealed that testing is not only performed by testing experts; employees in different organizational roles within the company have varying defect detection and fix rates. They concluded that it is crucial to understand the diversity of individuals involved in software testing from the perspective of end-users and the relevance of validation. They also provided their recommendations. This article has provided me with a more comprehensive understanding of software testing in real-world companies. Furthermore, it has also helped me understand why ScreenTracerViewer chose to use structured natural language descriptions.

2.1.6 Stakeholder participation in the development of an electronic medical record system in Malawi

In this paper, Chawani et al. [6] focus on the involvement of stakeholders in the development of an Electronic Medical Record (EMR) system for health facilities in Malawi, Africa. They conduct a study on the diverse roles and forms of engagement among these stakeholders. The paper illustrates how participation evolves over time and with the progress of the project, and it also highlights the challenges faced by stakeholders in participating in impoverished environments. This paper has provided me with insights into the varying situations of stakeholders in different environments.

2.2 BDD: Behavior-Driven Development

In software engineering, Behavior-Driven Development (BDD) is a software development process that aligns well with the agile software development approach. It promotes collaboration among development team (developers and testers), product managers, and customers representatives within a software project. BDD was evolved from Test-Driven Development (TDD). This methodology combines the fundamental principles of TDD with concepts from domain-driven design and object-oriented analysis and design, offering software development and management teams a shared framework and tools for collaborative software development. BDD leverages simple formatted natural language (e.g., Gherkin) to enhance communication between development teams, product managers and customers. This facilitates a better understanding of business objectives by the development team, allowing them to align their efforts with the product requirements defined by product managers.

2.3 SharpAvi

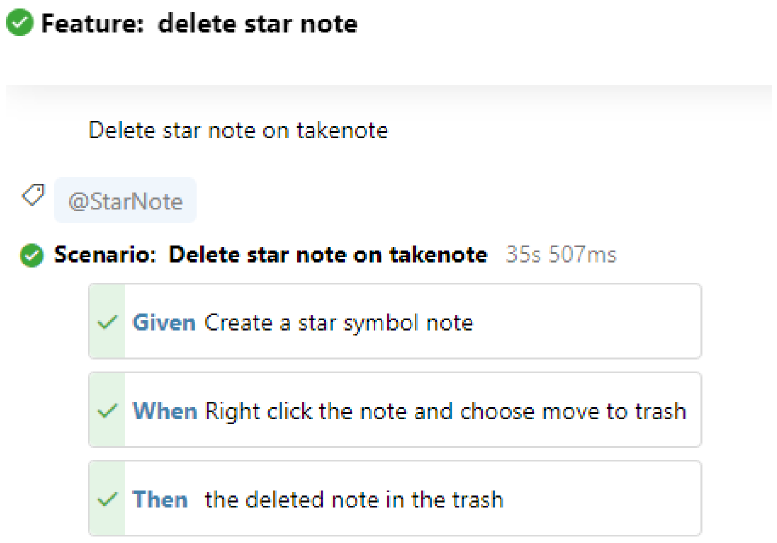
SharpAvi is a simple .NET library designed for the generation of AVI format video files. It specializes in rendering video sequences without relying on native APIs like DirectShow or external command-line utilities such as FFmpeg. This library doesn't require any external dependencies. It's entirely built using pure .NET code. The resulting files adhere to the OpenDML extensions, allowing for nearly unlimited file sizes, eliminating the previous 2GB limit. Creating a video with SharpAvi involves supplying individual in-memory bitmaps and audio samples. The library includes a few built-in encoders for both video and audio. Regardless of the specific codec used, the output format will always be AVI.

2.4 SpecFlow

SpecFlow is a test automation solution designed for .NET, following the principles of Behavior-Driven Development (BDD). It provides a platform to define, manage, and automatically execute human-readable acceptance tests within .NET projects, including both Full Framework and .NET Core applications. These tests are composed using Gherkin, a language that enables the creation of test cases in a natural and easily understandable manner. Furthermore, SpecFlow leverages the official Gherkin parser, supporting more than 70 languages, ensuring versatility and accessibility across different linguistic contexts. Additionally, SpecFlow has the capability to generate LivingDoc, a feature that simplifies the process of validating whether the specified requirements have been met. This functionality is valuable to both technical and non-technical team members, facilitating a more intuitive assessment of compliance with the specified requirements.

2.5 LivingDoc

LivingDoc is an extension to the SpecFlow framework. SpecFlow automatically generates validated specification scenarios as living documentation (LivingDoc). LivingDoc showcases Gherkin feature specifications and their automated validation results. This documentation can be effortlessly shared with stakeholders or product managers who may not possess a understanding of software programming, facilitating effective communication and collaboration within the development process. An example can be found in Figure2.1.



✓ **Feature: delete star note**

Delete star note on takenote

@StarNote

✓ **Scenario: Delete star note on takenote** 35s 507ms

✓ **Given** Create a star symbol note

✓ **When** Right click the note and choose move to trash

✓ **Then** the deleted note in the trash

Figure 2.1: One sample of LivingDoc

LivingDoc simplifies the process of understanding whether the requirements have been met for stakeholders or product managers in real-time. It provides a straightforward and intuitive way for them to assess the alignment of the software with the specified requirements.

2.6 Selenium IDE

Selenium IDE is a browser extension available for Google Chrome, Mozilla Firefox, and Microsoft Edge. It serves the purpose of recording and reproducing actions of the user within the browser. Selenium IDE records these actions as test cases using existing Selenium commands, with parameters defined by the context of each element. Users can subsequently replay these tests within the IDE by selecting the desired test. The desired test is played back in the browser and the user can see a step-by-step description of actions while watching the video playback. Selenium IDE also offers a valuable feature: the ability to export a test to WebDriver code. This WebDriver code can be used with ScreenTracer.

2.7 ScreenTracer

ScreenTracer is a project by Holzmann [7]. This software comprises two components: ScreenTracer and ScreenTracerViewer. The ScreenTracerViewer here refers to the original version of ScreenTracerViewer. ScreenTracer is responsible for recording Selenium test cases and saving them in ".stc" format files, while ScreenTracerViewer plays ".stc" format files as videos. ScreenTracerViewer provides a range of useful features to users. It allows users to select which videos to play and provides play and stop buttons for video playback control. Additionally, ScreenTracerViewer includes previous and next screen buttons, enabling users to switch between different videos in the playlist. While a video is playing, a video progress bar displays the current time position within the video. Moreover, users have the option to adjust the playback speed using a slider. A more comprehensive explanation can be found in a paper authored by Holzmann et al. [8].

Chapter 3

Requirements

This chapter explains both the mandatory and optional requirements. The problems which lead to each requirement are to be clarified. In Section 3.1 the design process is further explained. Section 3.2 states the stakeholders. Section 3.3 lists the mandatory requirements, and Section 3.4 lists the optional requirements. The template named "FunctionalMASTER" used for formulating requirements is from Rupp et al. [9].

3.1 Design process

The design process follows the agile methodology. Once the requirements are gathered, the project is structured into four milestones, which are managed using the Trello application — a Kanban board tool. Each milestone consists of various subtasks, each having its own set deadlines. These subtasks are represented as cards on a Kanban board within Trello. A weekly meeting is conducted with the supervisor to evaluate the progress made during the week, address any encountered challenges, and brainstorm ideas for new tasks in the upcoming week. These new tasks are subsequently added to the Trello Kanban board. This systematic approach ensures high productivity and allows for the measurement of work progress on a weekly basis. To maintain consistent and high-quality code, the project adheres to the Microsoft C# Coding convention.

3.2 Stakeholders

Stakeholders encompass all members within the development team, including software developers and product managers, as well as external customers. Leveraging video files and Gherkin specifications, developers can conveniently showcase the software product developed in accordance with the requirements of customers. Simultaneously, product managers and customers can intuitively verify whether the software product complies with the specified requirements.

3.3 Mandatory Requirements

R1 When the user ends the screen recording, ScreenTracer shall be able to generate a video and automatically store the video in .AVI format.

R2 When the user ends the screen recording, ScreenTracer shall be able to automatically store a link between time periods of the video and the Gherkin specification in a file (e.g. json, csv).

R3 When a video file is playing in ScreenTracerViewer, ScreenTracerViewer shall provide the user with the ability to also see the Gherkin specification of this video (Gherkin specification uses the Given-When-Then grammar to describe what is happening in the video).

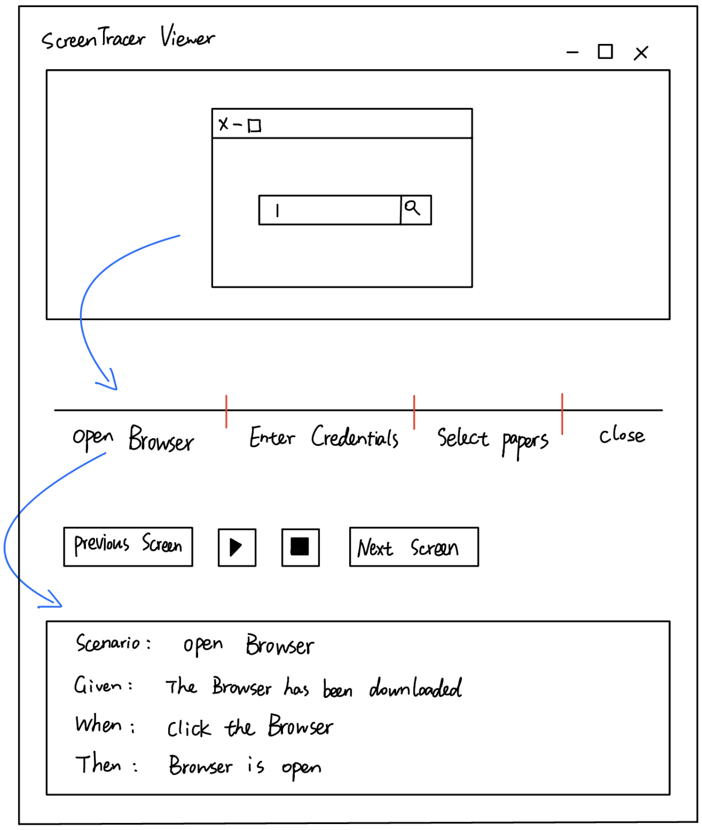


Figure 3.1: Gherkin specification of the video

R4 When multiple video files are open in ScreenTracerViewer, ScreenTracerViewer shall separate the multiple videos by a thick vertical line in the progress bar and provide the user with the ability to play multiple videos under one view in ScreenTracerViewer.

R5 When multiple video files are open in ScreenTracerViewer, ScreenTracerViewer shall provide the user with the ability to define the playback order of videos.

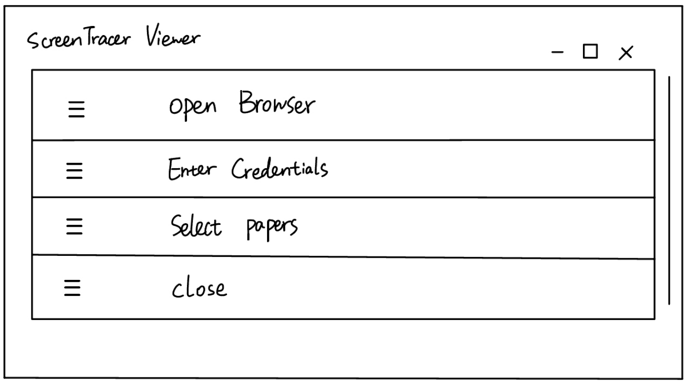


Figure 3.2: Playback order of videos

3.4 Optional Requirements

OR1 SharpAvi is replaced with an MP4 package.

OR2 When the user hovers the mouse over the label of the video in tab bar of ScreenTracerViewer, ScreenTracerViewer shall be able to display the full name of the video.

OR3 When ScreenTracer executes an executable file, ScreenTracer shall be able to modify the name of the obtained executable file and use the modified name as the name for the generated video file.

Chapter 4

Implementation

This chapter explains the process, methods and ideas of implementing the requirements in detail.

4.1 R1: When the user ends the screen recording, ScreenTracer shall be able to generate a video and automatically store the video in .AVI format.

To enhance the video recording and generation capabilities of ScreenTracer, the transition to a more reliable driver was undertaken. The previous performance shortcomings of driver, characterized by irregularities and choppiness, necessitated this shift. The objective was to ensure the production of videos of superior quality. The replacement driver was chosen after careful consideration, with SharpAvi emerging as the preferred solution. The primary driver behind this decision was the seamless integration of the capabilities of SharpAvi to enable automatic video storage in the .AVI format. This transition was executed through a structured approach. The initial step involved the integration of the SharpAvi package into the Visual Studio environment. Subsequently, existing functionalities within ScreenTracer were adapted and harmonized with features of SharpAvi. Notably, functions like startRecording() and stopRecording() underwent restructuring to ensure a cohesive interaction between ScreenTracer and SharpAvi.

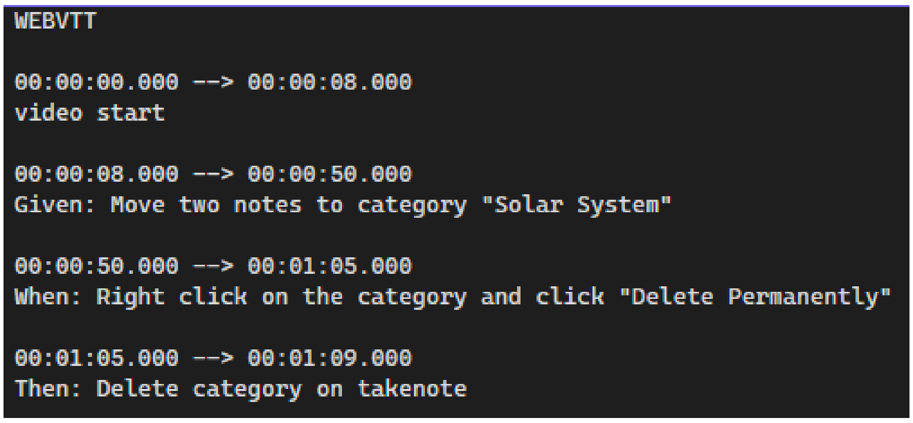
```
public void StartRecording()
       {
           try
           {
               MessageBox.Show("start recording");
               /*Elapsed = "00:00";
```

```
                recordingTimer.Start();*/
                string exePath = goalTestExeLoc;
                string processName =
                    Path.GetFileNameWithoutExtension(exePath);
                lastFileName = System.IO.Path.Combine(saveDirectory,
                    processName + ".avi");
                //lastFileName =
                    System.IO.Path.Combine(saveDirectory,
                    ScenarioGetter() + ".avi"

                //recordingArea = w1.GetArea();
                recorder = new Recorder(lastFileName, encoder,
                    encodingQuality, recordingArea);
                recordingStopwatch.Start();
            }
            catch (Exception ex)
            {
                StopRecording();
                MessageBox.Show("error:" + ex.Message);
            }
        }
```

4.2 R2: When the user ends the screen recording, ScreenTracer shall be able to automatically store a link between time periods of the video and the Gherkin specification in a file (e.g. json, csv).

In the previous version of ScreenTracer, its functionality was restricted to recording and generating videos. However, an extended capability is now being pursued, aiming to establish an automated connection between specific time intervals in the video and the corresponding Gherkin specification. This link will be encapsulated within a designated file. To achieve this, the initial task involves determining the appropriate file format to accommodate this association. Several options, such as JSON, CSV, and WebVTT, are under consideration. Given the overarching goal of seamlessly integrating this file with the existing HTML player, the preference has been given to the WebVTT format. The WebVTT format has emerged as the most suitable choice due to its alignment with HTML requirements and its inherent simplicity. This format incorporates both timestamps and subtitles. Timestamps enable the indication of specific temporal segments within the video, while subtitles effectively describe the corresponding Gherkin specification. With the file format determined, the subsequent step involves devising an approach to combine time intervals and Gherkin specifications into a WebVTT file. The specific format for time intervals and Gherkin specifications in the WebVTT file can be found in Figure4.1.



```
WEBVTT

00:00:00.000 --> 00:00:08.000
video start

00:00:08.000 --> 00:00:50.000
Given: Move two notes to category "Solar System"

00:00:50.000 --> 00:01:05.000
When: Right click on the category and click "Delete Permanently"

00:01:05.000 --> 00:01:09.000
Then: Delete category on takenote
```

Figure 4.1: Time intervals and Gherkin specifications are stored in a WebVTT file

Upon analysis, it has been noted that during the execution of the Selenium exported .exe file, a series of outputs are displayed within the console. These outputs provide a detailed account of the unfolding events within the video. By capturing these outputs and utilizing the time-retrieval function provided by SharpAvi, ScreenTracer can adeptly integrate both elements within a WebVTT file. This strategy effectively establishes a coherent connection between the temporal aspects of the video and the corresponding Gherkin specification.

```
private Task<int> RunProcessAsync(Process process)
       {
           var task = new TaskCompletionSource<int>();

           List<string> outputLines = new List<string>();

           string videoName = "";

           process.Start();

           outputLines.Add((recordingStopwatch.StartAt -
               recordingStopwatch.StartAt) + " - " + "video start");

           while (!process.StandardOutput.EndOfStream)
           {
               string line = process.StandardOutput.ReadLine();

               if (line.Contains("Given") || line.Contains("When")
                    || line.Contains("Then"))
                {
                    timeStopwatch.Start();
                    outputLines.Add((timeStopwatch.StartAt -
                        recordingStopwatch.StartAt) + " - " + line);

                }

                if (line.Contains("Scenario"))
                {
                    videoName += line.Replace("Scenario: ", "");
                }
            }

            outputLines.Add(recordingStopwatch.Elapsed + " - " +
                "video is over");
            process.WaitForExit();

            List<string> webvttTimestamps =
                ConvertToWebVTTTimestamps(outputLines);
            List<string> subtitleTexts =
                GetSubtitleTexts(outputLines);
            string webvttContent =
                BuildWebVTTContent(webvttTimestamps, subtitleTexts);
            string exePath = goalTestExeLoc;
            string processName =
                Path.GetFileNameWithoutExtension(exePath);
            string filePath = saveDirectory + "\\" + videoName +
                ".vtt";
            System.IO.File.WriteAllText(filePath, webvttContent);

            process.BeginErrorReadLine();

            return task.Task;
        }
```

4.3 R3: When a video file is playing in ScreenTracerViewer, ScreenTracerViewer shall provide the user with the ability to also see the Gherkin specification of this video (Gherkin specification uses the Given-When-Then grammar to describe what is happening in the video).

To meet this requirement, a feature was integrated into ScreenTracerViewer to enable user selection of video files from their local directories. Concurrently, a webvtt file bearing the same name as the video file is automatically identified within the corresponding directory. Subsequently, ScreenTracerViewer displays this associated webvtt file as subtitles below the video playback. The objective of this enhancement is to provide users with direct access to the Gherkin specification aligned with the video content.

```
function loadVideoAndSubtitles() {
  var fileInput = document.getElementById('videoInput');
  var videoFiles = fileInput.files;

  if (videoFiles.length > 0) {
    videosContainer.innerHTML = "";
    tabContainer.innerHTML = "";
    videoData = [];

    for (var i = 0; i < videoFiles.length; i++) {
      var videoFile = videoFiles[i];
      var objectURL = URL.createObjectURL(videoFile);

      loadSubtitles(videoFile, objectURL, i);
    }
  }
}
```

This implementation involves storing the Gherkin specification alongside corresponding timestamps within the webvtt file. The process commences with the conversion of the Selenium-generated code into an executable SpecFlow project. The procedural details for this conversion are expounded upon in subsection 4.3.1, titled "How to convert scenario files into test codes." Following this, the approach outlined in R2 is employed to retrieve outputs from the executable SpecFlow project. Specifically, this involves procuring the Gherkin specification. Subsequently, this specification is harmonized with pertinent timestamps and deposited within the webvtt file.

4.3.1 How to convert scenario files into test codes

In the software development process of BDD, developers need to convert scenarios files into executable automated test codes. Because of the use of selenium in our project, it is different from the general use of BDD to develop software products. The flow chart of implementation can be found in Figure4.2.

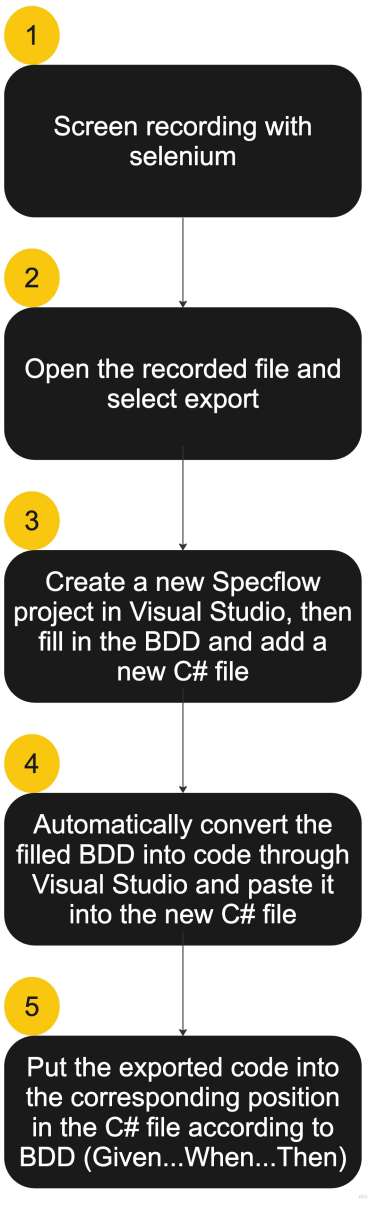

1
Screen recording with selenium
2
Open the recorded file and select export
3
Create a new Specflow project in Visual Studio, then fill in the BDD and add a new C# file
4
Automatically convert the filled BDD into code through Visual Studio and paste it into the new C# file
5
Put the exported code into the corresponding position in the C# file according to BDD (Given...When...Then)

Figure 4.2: Process of converting scenario files into test codes

(1.) To initiate the process, Selenium is employed for screen recording and the selection of the storage location of the file generated by Selenium. The generated file format is ".side."

(2.) After the video recording is completed, developers open the ".side" file and replay it with the Selenium. Within the Selenium interface, a button with three dots can be found at the top-left corner. This button provides the functionality to automatically convert actions of the recorded video into C# code and export it. Subsequently, developers click on this button and select the export location.

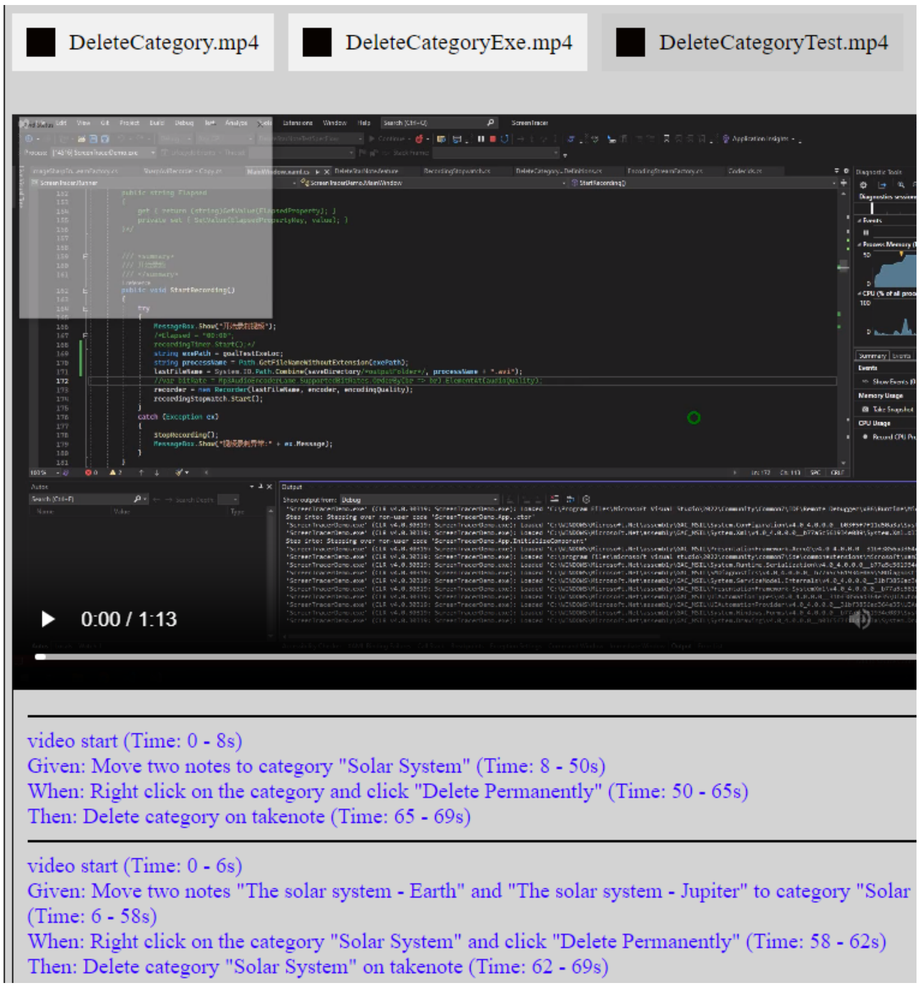
(3.) Now, developers create a new project in Visual Studio and search for "Specflow Project" in the search bar, selecting it. This action triggers the automatic generation of several folders in the Solution Explorer on the right side of the Visual Studio interface. The folders of interest are "Features" and "StepDefinitions". Developers right-click the "Features" folder and select "New Item". An interface appears for selection and search. Developers click on "Specflow" on the left side of this interface and choose "Feature File for Specflow". This results in the creation of a new ".feature" file under the "Features" folder, automatically generating "Scenario" and "Given...When...Then" (Gherkin) elements. Developers then populate this file with the software product usage scenario.

(4.) Next, developers create a new empty C# file within the "StepDefinitions" folder. They right-click on the populated ".feature" file and select "Define Steps". Next, the developer clicks "Copy to clipboard" on the appear interface. This action automatically converts the content of the filled ".feature" file (BDD) into executable automated test codes, which are then pasted into the new, empty C# file under the "StepDefinitions" folder.

(5.) Finally, developers integrate the code exported by Selenium into the corresponding positions, aligning with the "Scenario...Given...When...Then" structure in the C# file. Through these outlined steps, developers seamlessly combine the code exported by Selenium with the BDD framework, Specflow.

4.4 R4: When multiple video files are open in ScreenTracerViewer, ScreenTracerViewer shall separate the multiple videos by a thick vertical line in the progress bar and provide the user with the ability to play multiple videos under one view in ScreenTracerViewer.

The outlined plan has been realized in an alternative manner. In order to facilitate user-friendly video selection within the ScreenTracerViewer, the decision was made to enable the playback of multiple videos in a unified view. Rather than amalgamating multiple videos into a single entity separated by a distinct vertical line in the progress bar, a more intuitive approach was devised. The solution involved the creation of a tab bar, with each tab corresponding to an individual video. By clicking on a specific tab, users are able to seamlessly transition between videos within the view. This design maintains the capacity to play multiple videos within a single view, yet the navigation process is markedly more streamlined. The final implementation style of ScreenTracerViewer can be seen in Figure4.3.



video start (Time: 0 - 8s)
Given: Move two notes to category "Solar System" (Time: 8 - 50s)
When: Right click on the category and click "Delete Permanently" (Time: 50 - 65s)
Then: Delete category on takenote (Time: 65 - 69s)

video start (Time: 0 - 6s)
Given: Move two notes "The solar system - Earth" and "The solar system - Jupiter" to category "Solar
(Time: 6 - 58s)
When: Right click on the category "Solar System" and click "Delete Permanently" (Time: 58 - 62s)
Then: Delete category "Solar System" on takenote (Time: 62 - 69s)

Figure 4.3: ScreenTracerViewer

4.5 R5: When multiple video files are open in ScreenTracerViewer, ScreenTracerViewer shall provide the user with the ability to define the playback order of videos.

To enhance the user experience, ScreenTracerViewer has been equipped with a feature that facilitates the reordering of videos. This functionality incorporates tab icons and interactive elements, enabling effortless dragging and swapping of tabs. This allows users to adjust the sequence of video content according to their preferences, promoting a more personalized and intuitive interaction within the ScreenTracerViewer.

```
function doDragging(event) {
  if (isDragging) {
    var tabs = document.querySelectorAll(".tab-icon");
    for (var i = 0; i < tabs.length; i++) {
      tabs[i].style.cursor = "grabbing";
    }
    ...
    if (newIndex !== currentIndex) {
      swapTabs(currentIndex, newIndex);
      activeTabIndex = newIndex;
    }
    dragEndIndex = newIndex;
  }
}

function swapTabs(index1, index2) {
  var tabs = document.querySelectorAll(".tab");
  var temp = tabs[index1].innerHTML;
  tabs[index1].innerHTML = tabs[index2].innerHTML;
  tabs[index2].innerHTML = temp;
}

function reorderVideos(index1, index2) {
  var tempVideo = videoData[index1];
  var tempTracks = tracks[index1];

  videoData[index1] = videoData[index2];
  tracks[index1] = tracks[index2];

  videoData[index2] = tempVideo;
  tracks[index2] = tempTracks;

}
```

Chapter 5

Evaluation and Discussion

In this chapter, the results of the work are presented and described in Test Cases. Because the main achievement is to add a lot of functions that ScreenTracer and ScreenTracerViewer did not have before and most of the tasks are very technical, so the test case is used as the evaluation. In addition to the evaluation, this chapter also demonstrates how ScreenTracer and ScreenTracerViewer are applied in practical software development through a specific example. This specific example is delved into to derive more related content. This part of the content serves as a discussion.

5.1 Test Environment

Operating System: [Windows 10 Pro]

SharpAvi: [3.0.1]

.NET: [6.0 and 4.8]

SpecFlow: [3.9.74]

Selenium.WebDriver: [4.10.0]

5.2 Test Case

This section lists all the test cases of this project.

Test Case Title:	ScreenTracer Video Generation
Test Case ID:	TC-001
Objective:	To verify that ScreenTracer can successfully generate a video upon the user ending the screen recording.
Preconditions:	1. ScreenTracer application is installed and functional. 2. User has initiated a screen recording session.
Test Steps:	1. Start the screen recording using ScreenTracer. 2. Simulate the user's actions as if ending the screen recording session.
Expected Results:	1. Upon the user ending the recording session, ScreenTracer initiates the video generation and saving process automatically.
Test Data:	Video content of various screen activities.

Table 5.1: ScreenTracer shall be able to automatically generate a video.

Test Case Title:	ScreenTracer Automated Storage of Video in .AVI Format
Test Case ID:	TC-002
Objective:	To verify that ScreenTracer can automatically store the video in the .AVI format upon the user ending the screen recording.
Preconditions:	1. ScreenTracer application is installed and functional. 2. User has initiated a screen recording session.
Test Steps:	1. Start the screen recording using ScreenTracer. 2. Simulate the user's actions as if ending the screen recording session.
Expected Results:	1. The generated video is automatically saved in .AVI format in the specified directory or default location.
Test Data:	Video content of various screen activities.

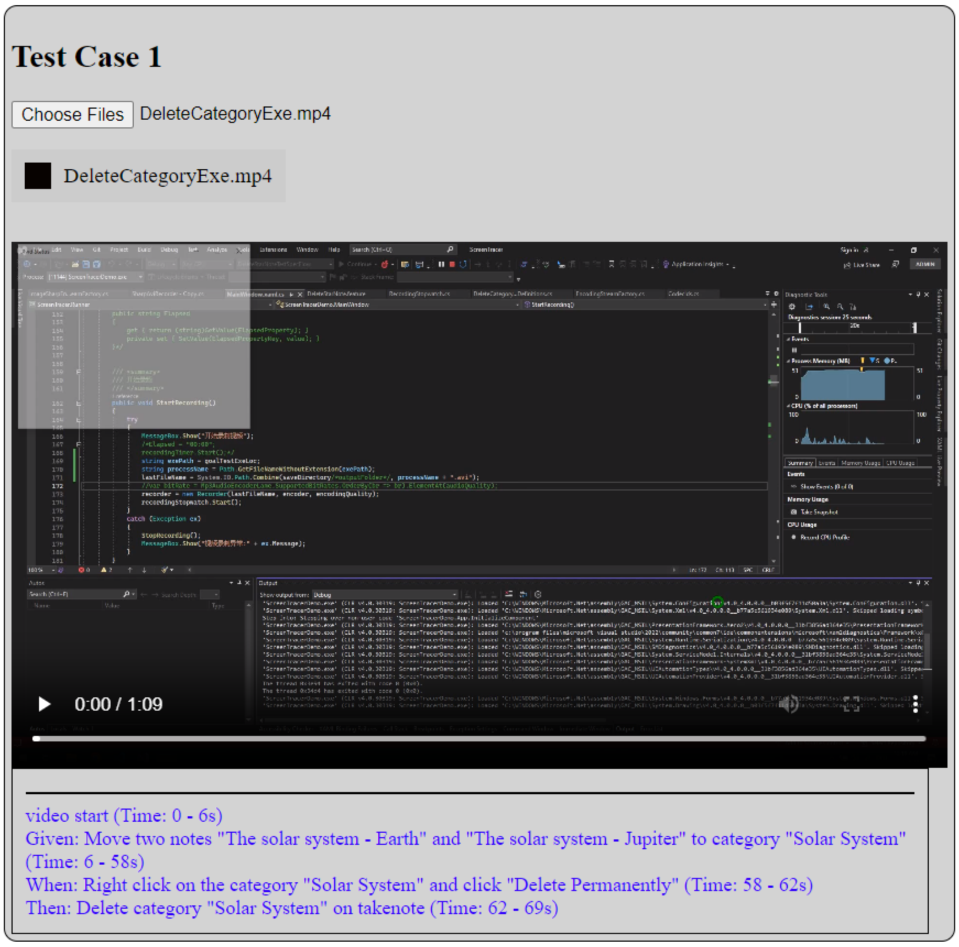
Table 5.2: ScreenTracer shall be able to automatically store the video in .AVI format.

Test Case Title:	Automatic Link Storage between Video Time Periods and Gherkin Specifications
Test Case ID:	TC-003
Objective:	To verify that ScreenTracer automatically stores a link between time periods of the recorded video and their corresponding Gherkin specifications in a WebVTT file.
Preconditions:	1. ScreenTracer application is installed and functional. 2. Executable file under SpecFlow framework is available. 3. User has initiated a screen recording session.
Test Steps:	1. Start the screen recording using ScreenTracer. 2. Simulate the user's actions as if ending the screen recording session. 3. Wait for the video generation and saving process to complete.
Expected Results:	1. Time intervals and Gherkin specifications are successfully stored in a WebVTT file.
Test Data:	Test executable file under SpecFlow framework.

Table 5.3: ScreenTracer shall be able to automatically store a link between time periods of the video and the Gherkin specification in a file (e.g. json, csv).

Test Case Title:	Display Gherkin Specification for Playing Video in ScreenTracerViewer
Test Case ID:	TC-004
Objective:	To verify that ScreenTracerViewer provides users with the ability to see the Gherkin specification of a playing video.
Preconditions:	1. ScreenTracerViewer application is installed and functional. 2. A valid video file is available. 3. A WebVTT file with the same name and path as the video file is available.
Test Steps:	1. Launch the ScreenTracerViewer application. 2. Select a video file. 3. Start playing the selected video.
Expected Results:	1. During video playback, the Gherkin specification of the playing video is displayed in a designated area.
Test Data:	A video file and a WebVTT file with the same name and path as the video file.

Table 5.4: ScreenTracerViewer shall provide the user with the ability to also see the Gherkin specification of this video (Gherkin specification uses the Given-When-Then grammar to describe what is happening in the video).



Test Case 1

Choose Files DeleteCategoryExe.mp4

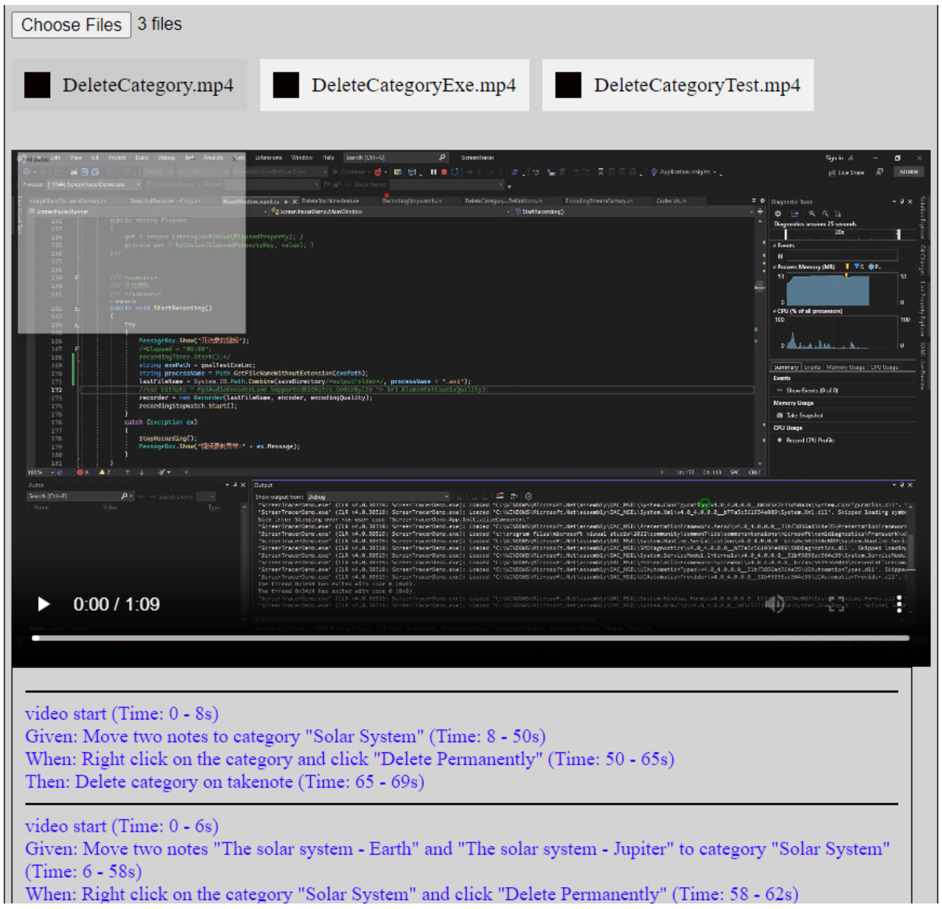
DeleteCategoryExe.mp4

video start (Time: 0 - 6s)
Given: Move two notes "The solar system - Earth" and "The solar system - Jupiter" to category "Solar System" (Time: 6 - 58s)
When: Right click on the category "Solar System" and click "Delete Permanently" (Time: 58 - 62s)
Then: Delete category "Solar System" on takenote (Time: 62 - 69s)

Figure 5.1: The Gherkin specification of the video

Test Case Title:	Play Multiple Videos under one view in ScreenTracerViewer
Test Case ID:	TC-005
Objective:	To verify that ScreenTracerViewer allows users to play multiple videos under one view.
Preconditions:	1. ScreenTracerViewer application is installed and functional. 2. Multiple video files and corresponding WebVTT files are available for testing.
Test Steps:	1. Launch the ScreenTracerViewer application. 2. Select multiple video files. 3. Click on tab-icons of the different videos and play videos.
Expected Results:	1. Multiple Videos are played under the same view.
Test Data:	Multiple video files.

Table 5.5: ScreenTracerViewer shall provide the user with the ability to play multiple videos under one view in ScreenTracerViewer.



video start (Time: 0 - 8s)
Given: Move two notes to category "Solar System" (Time: 8 - 50s)
When: Right click on the category and click "Delete Permanently" (Time: 50 - 65s)
Then: Delete category on takenote (Time: 65 - 69s)

video start (Time: 0 - 6s)
Given: Move two notes "The solar system - Earth" and "The solar system - Jupiter" to category "Solar System" (Time: 6 - 58s)
When: Right click on the category "Solar System" and click "Delete Permanently" (Time: 58 - 62s)

Figure 5.2: Multiple videos under one view

Test Case Title:	Select the video to play in ScreenTracerViewer
Test Case ID:	TC-006
Objective:	To verify that ScreenTracerViewer allows users to select which video to play.
Preconditions:	1. ScreenTracerViewer application is installed and functional. 2. Multiple video files and corresponding WebVTT files are available for testing.
Test Steps:	1. Launch the ScreenTracerViewer application. 2. Select multiple video files. 3. Click on tab-icons of the different videos.
Expected Results:	1. When the user clicks on the tab, the video within the view jumps to the video corresponding to the tab.
Test Data:	Multiple video files.

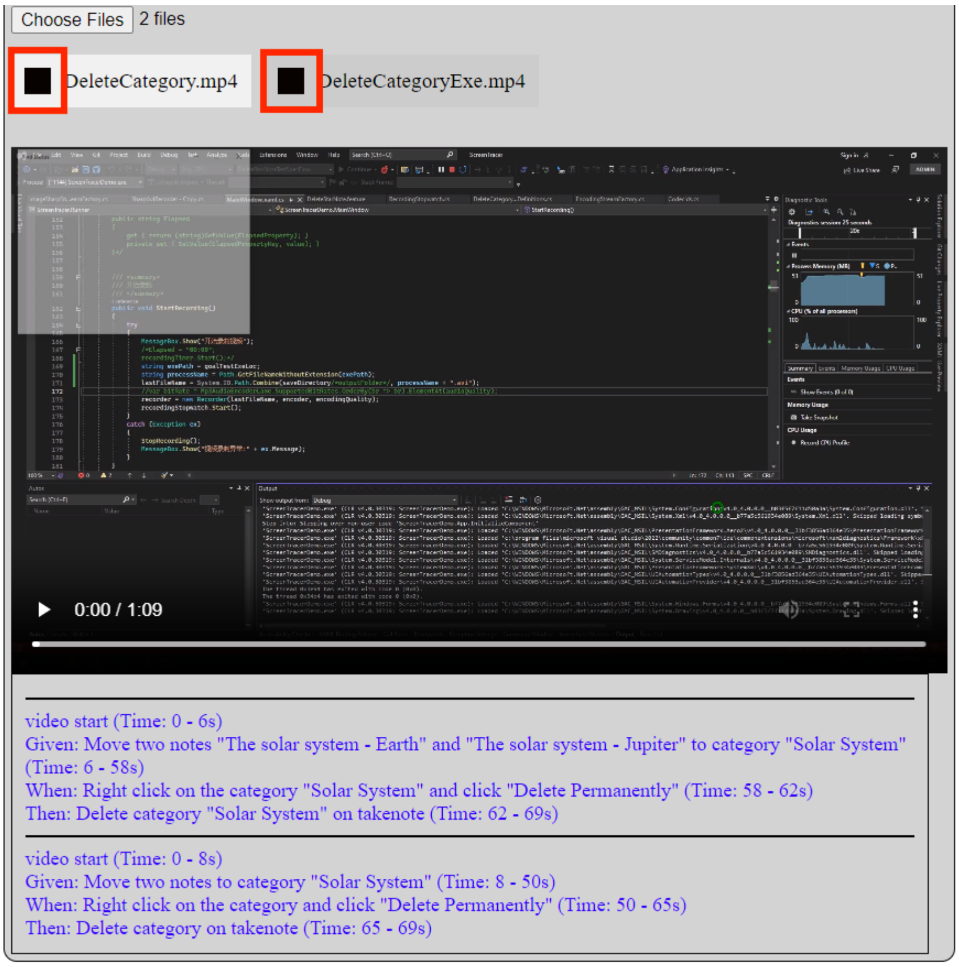
Table 5.6: ScreenTracerViewer shall provide the user with the ability to select which video to play in ScreenTracerViewer.

Test Case Title:	Separate the Gherkin specifications by a horizontal line in ScreenTracerViewer
Test Case ID:	TC-007
Objective:	To verify that ScreenTracerViewer separates the Gherkin specifications of different videos by a horizontal line.
Preconditions:	1. ScreenTracerViewer application is installed and functional. 2. Multiple video files and corresponding WebVTT files are available for testing.
Test Steps:	1. Launch the ScreenTracerViewer application. 2. Select multiple video files.
Expected Results:	1. The Gherkin specifications of all videos are displayed in the designated area below the video, and each video Gherkin specification is separated by a horizontal line.
Test Data:	Multiple video files.

Table 5.7: ScreenTracerViewer shall separate the Gherkin specifications of different videos by a horizontal line.

Test Case Title:	Define Playback Order of Videos in ScreenTracerViewer
Test Case ID:	TC-008
Objective:	To verify that ScreenTracerViewer allows users to define the playback order of multiple video files.
Preconditions:	1. ScreenTracerViewer application is installed and functional. 2. Multiple video files are available for testing.
Test Steps:	1. Launch the ScreenTracerViewer application. 2. Select multiple video files. 3. Drag and drop tab-icons of the different videos.
Expected Results:	1. The order of videos and tabs changes according to the drag and drop of tabs.
Test Data:	Multiple video files.

Table 5.8: ScreenTracerViewer shall provide the user with the ability to define the playback order of videos.



video start (Time: 0 - 6s)
Given: Move two notes "The solar system - Earth" and "The solar system - Jupiter" to category "Solar System" (Time: 6 - 58s)
When: Right click on the category "Solar System" and click "Delete Permanently" (Time: 58 - 62s)
Then: Delete category "Solar System" on takenote (Time: 62 - 69s)

video start (Time: 0 - 8s)
Given: Move two notes to category "Solar System" (Time: 8 - 50s)
When: Right click on the category and click "Delete Permanently" (Time: 50 - 65s)
Then: Delete category on takenote (Time: 65 - 69s)

Figure 5.3: Define the playback order of videos

5.3 Test Results

After testing, this project has achieved the expected results for all test cases. The only issue arose during the testing of TC-001 (Test Case ID) when the user attempted to select a specific area on the screen for recording, ScreenTracer exhibited instability. This instability is related to factors such as computer configuration, codec settings, frame rate, and more, indicating a need for further improvement. However, when the user recorded the entire screen, ScreenTracer performed very stably and met the expected results of TC-001. Since the requirements of the project were focused on recording videos and automatically generating video files, without explicit demands for allowing the user to select screen regions for recording, this project still achieved the expected results for TC-001. This demonstrates that the project has successfully passed all tests. The test video can be found on the accompanying CD attached to this thesis.

5.4 The Practical Use of ScreenTracer and ScreenTracerViewer

In this section, a highly realistic and concrete example will be presented to elucidate how ScreenTracer and ScreenTracerViewer assist the development team to work better. In this example, a software development company has been tasked with developing an official website application for an airline company. To ensure the website application aligns with the airline requirements of company, the software company has assigned its product manager to engage in meetings and discussions with the airline company, their customer. The airline requirements of company include the ability for users of the website to register as members, accumulate airline mileage points, redeem points for rewards, purchase tickets, cancel tickets, and check flight statuses and mileage points. The product manager conveys these requirements to the Business Analyst (BA) through user stories. Subsequently, the BA collaborates with the development team to analyze these user stories and distill them into specific usage scenarios, all of which are formatted using Gherkin syntax. For instance, for the scenario of a user purchasing a ticket, according to Gherkin format:

(1.) Given: The user is logged into their account.

(2.) When: The user selects a ticket, enters information, and clicks "Purchase."

(3.) Then: The website displays a successful ticket purchase message.

The development team proceeds to create the official website application of the airline company based on these usage scenarios. Once the source code is completed, they run it to launch the website application. Using Selenium, the development team records interactions with the website application, generating test code, and the test code is generated as an executable file. Next, the development team employs ScreenTracer to open the executable files. ScreenTracer records all interactions within the executable files, video and webvtt files are automatically generated after recording. At this point, the development team can hold meetings and discussions with the product manager and the customer. The development team uses ScreenTracerViewer to open the videos recorded by ScreenTracer. In interface of ScreenTracerViewer, both the video and its associated Gherkin specification are displayed. Product manager and customer can simultaneously view the video and its corresponding Gherkin specification. The video demonstrates various actions that can be performed on the website, such as purchasing tickets and checking flight statuses, while the Gherkin specification provides descriptions of these actions. Product manager and customer can verify whether the website application meets all of requirements of the airline company. If errors or improvements are needed, they can promptly provide feedback. The development team uses this feedback to enhance the website application, ultimately delivering a website application that fully aligns with the expectations of the airline company.

5.4.1 The benefits of using ScreenTracer and ScreenTracerViewer

From the example above, it is evident that ScreenTracer and ScreenTracerViewer provide benefits to everyone involved in the software project development process. For the development team, ScreenTracer and ScreenTracerViewer serve as valuable tools to ensure that software requirements are not overlooked and to prevent deviations from customer requirements that might arise from interpretations of developers during the development process. For the product manager and customer, ScreenTracer and ScreenTracerViewer provide a more convenient and intuitive way to verify whether the product meets customer requirements.

5.5 Software development process of BDD

The development process of the example presented in Section 5.4 follows the principles of Behavior-Driven Development (BDD), with the integration of ScreenTracer and ScreenTracerViewer. This section demonstrates the software development process of BDD. The software development flow chart of BDD can be seen in Figure5.4.

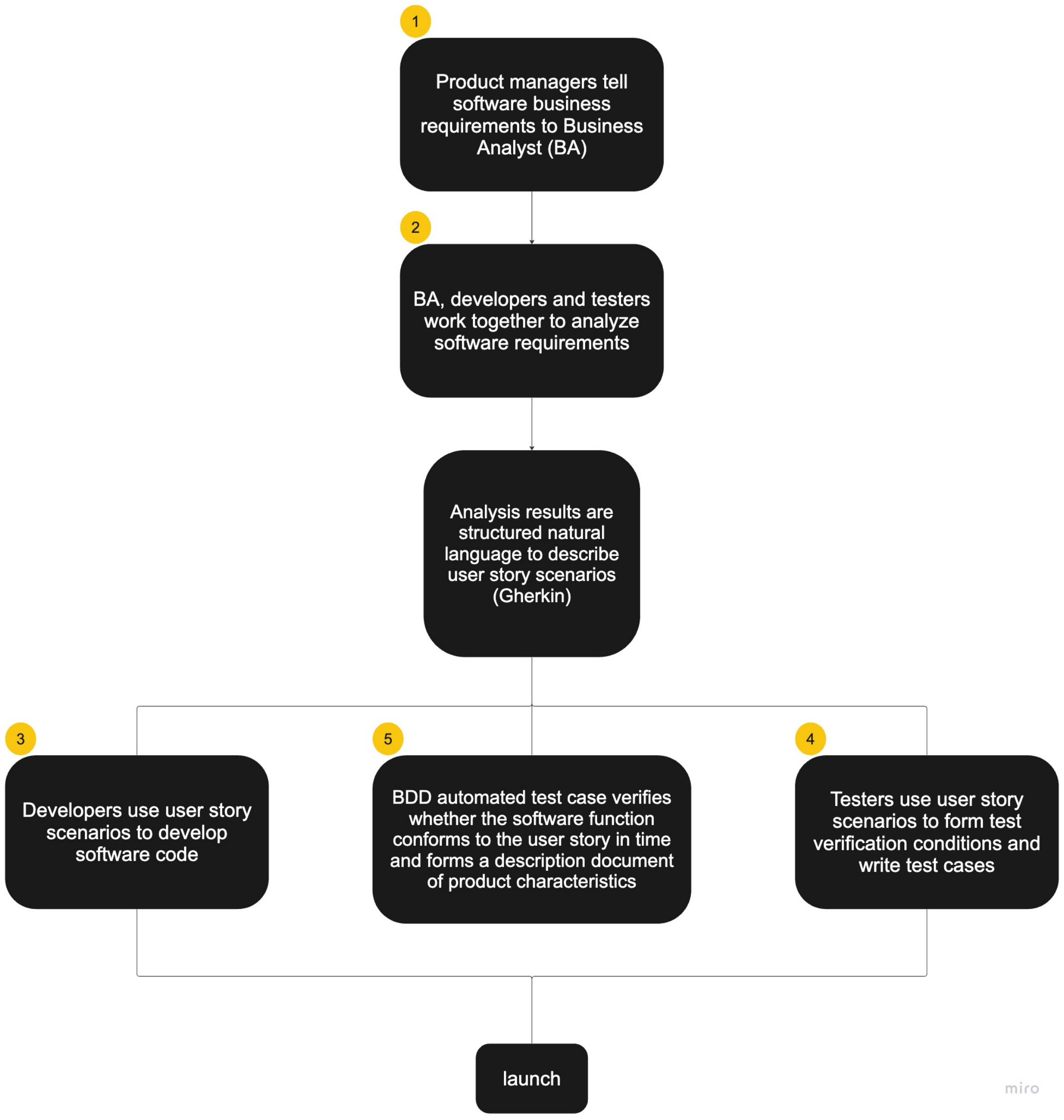

1
Product managers tell software business requirements to Business Analyst (BA)
2
BA, developers and testers work together to analyze software requirements
Analysis results are structured natural language to describe user story scenarios (Gherkin)
3
Developers use user story scenarios to develop software code
5
BDD automated test case verifies whether the software function conforms to the user story in time and forms a description document of product characteristics
4
Testers use user story scenarios to form test verification conditions and write test cases
launch
miro

Figure 5.4: Software development flow chart of BDD

BDD fosters close collaboration among all project members in software development: product managers, business analysts (BAs), software developers, and testers. To initiate the process, (1.) the product manager communicates the specific software product requirements to the Business Analyst through user stories. The primary advantage of utilizing user stories to outline software requirements at the outset of the project is that it avoids the potential for misunderstandings regarding software requirements among team members, such as BAs, and minimizes subjective interpretations. Once the business analyst comprehends the requirements of the software product, (2.) both the business analyst and the development team (comprising developers and testers) collaboratively analyze the user stories of the product manager. They collaboratively outline specific usage scenarios for the software product. These scenarios are described using structured, natural language with keywords, such as Gherkin. Subsequently, (3.) the development team leverages BDD tools to convert these scenario files into executable automated test codes. The procedural details for this conversion are expounded upon in Chapter 4, Section 4.3, Subsection 4.3.1, titled "How to convert scenario files into test codes." Developers write software code based on these usage scenarios and then execute the automated test codes to verify whether the developed software aligns with the acceptance criteria defined by the usage scenarios of the software product. Based on the results of automated testing, (4.) testers can perform manual testing and exploratory testing. Throughout the development process of the project, (5.) product managers can view the real-time automated test results produced by the software development team and the test reports generated by BDD tools. This ensures that the software implementation aligns with the specified software requirements. The aforementioned BDD development process demonstrates how BDD enhances collaboration among all project members. This collaborative approach empowers the development team to create software products that better meet customer requirements.

5.6 Gherkin and Test Case

In Section 5.5, it is mentioned that usage scenarios in the BDD development process are described using structured natural language Gherkin. This section introduces Gherkin and illustrates why using Gherkin is more convenient for testers in the development team, as compared to traditional Test Case. To foster improved team collaboration and streamline software development, structured natural language is employed for the articulation of customer software requirements. Gherkin, among the various structured languages utilized, emerges as a prominent choice. Gherkin is crafted to be non-technical and easily readable and comprehensible by individuals, regardless of their coding proficiency. It offers an effective means of describing software use cases that can be grasped by every team member. The typical syntax of Gherkin is Given-When-Then. Within this framework:

(1.) The Given step establishes the context or preconditions within the system, often referencing past events or conditions.

(2.) The When step articulates the specific actions to be executed.

(3.) The Then step outlines the expected outcomes stemming from the preceding actions, signifying the expected behavior or system state post-execution.

Incorporating Gherkin into the development process enhances clarity, facilitates collaboration, and ensures the effective communication and fulfillment of customer requirements. One sample of Gherkin can be found in Figure5.5.

```
Feature: Calculator

Calculator for adding two numbers

Scenario: Add two numbers
Add two numbers with the calculator
Given I have entered 50 into the calculator
And I have entered 70 into the calculator

When I press add
Then the result should be 120 on the screen
```

Figure 5.5: One sample of Gherkin

Test Case is a structured set of elements encompassing user inputs (in the form of test data), execution conditions, and expected outcomes. It is meticulously crafted to assess whether a specific software program aligns with and fulfills the stipulated customer requirements. Test Cases serve as a fundamental component of the software testing process, playing a pivotal role in ensuring the software's reliability, functionality, and compliance with customer expectations. The comparison between Gherkin and Test Case can be seen in Figure5.6.

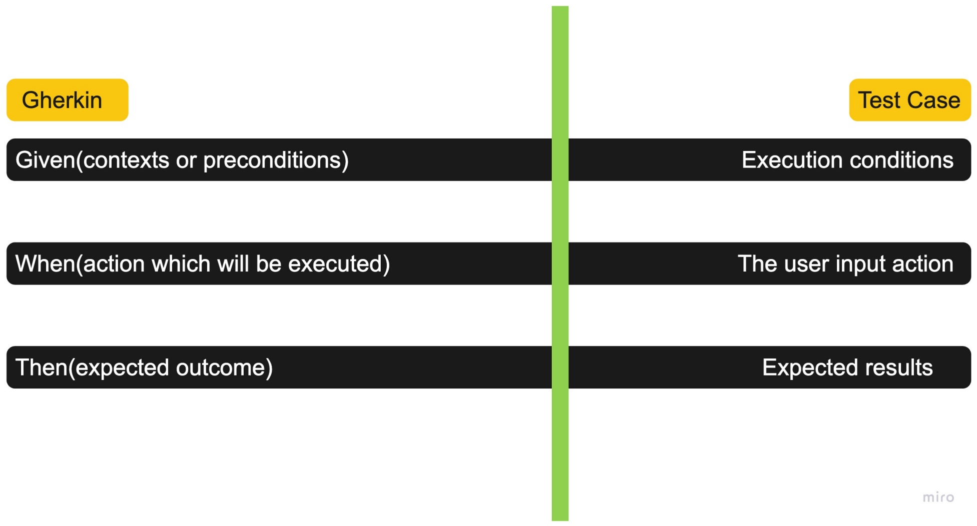


Figure 5.6: Gherkin and Test Case

The comparison chart highlights a one-to-one correspondence between the structure of Gherkin and that of a Test Case. Both Gherkin and Test Cases share a common composition, consisting of three key components: context, action, and expected results. This is the reason why Gherkin is more convenient for testers. Gherkin includes all the elements found in Test Case, and furthermore, it is more concise, clear, intuitive, and easier to understand.

5.7 The comparison between BDD- and traditional software development process

In Section 5.4, the development process of the example follows the principles of BDD, and Section 5.5 explains what BDD is.Then, this Section gives the advantages of BDD by comparing BDD with the traditional software development process. The traditional software development process can be seen in Figure5.7.

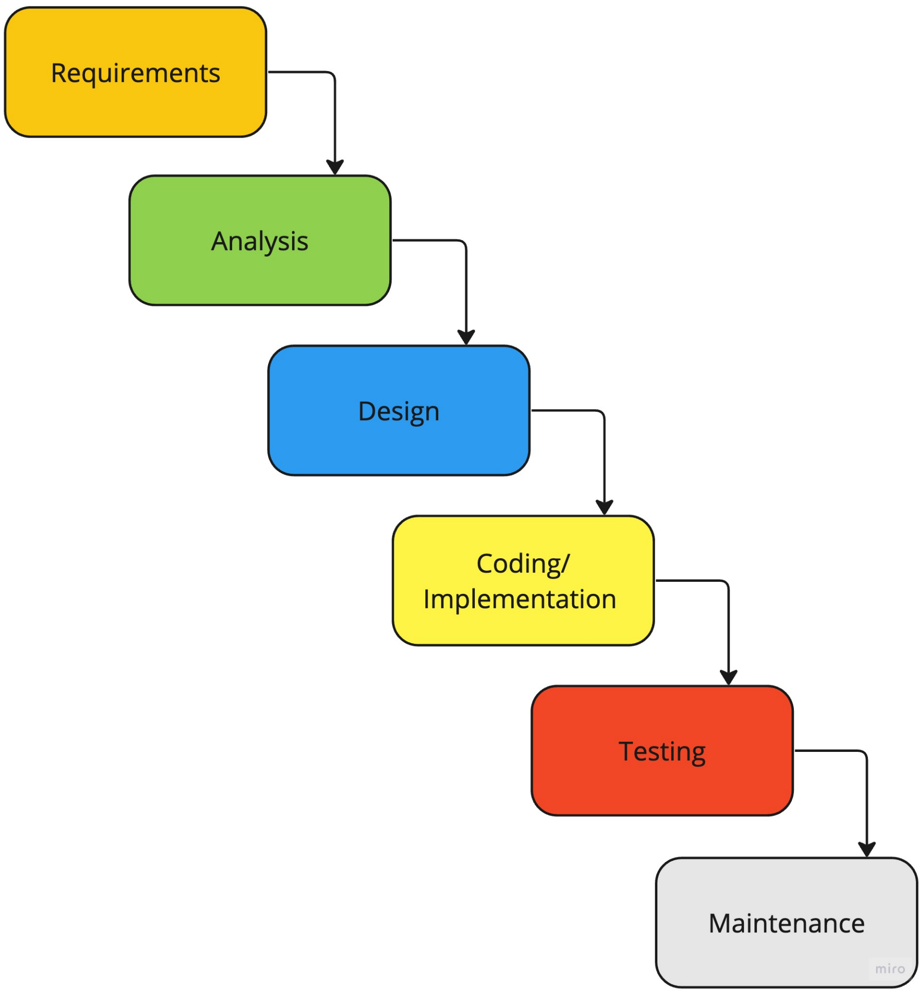


Figure 5.7: Traditional software development process

The initial stage of the traditional software development process involves the product manager conveying the desired software product requirements to software requirements analysts. Subsequently, these analysts document these requirements and produce software product requirements specifications. Based on these specifications, software developers proceed to write code and perform unit testing. Software testers then analyze the testing requirements as outlined in the software product requirements specifications, create test cases (use cases), and employ these test cases to evaluate the software products. Finally, the software development team generates functional and technical description documents for the software product. This constitutes the fundamental approach in traditional software development. However, the traditional software development model has inherent issues. During the transition from business and software requirements to coding, and onward to software testing, different roles and team members handle the necessary information at different stages. This introduces numerous opportunities for information loss, errors, or even the inadvertent neglect of the original customer requirements. Any misstep in this process can result in challenges for the software development team in delivering products that align with customer expectations and deadlines. In contrast, Behavior Driven Development (BDD) offers effective solutions to these issues. BDD, an agile software development methodology, actively involves all members of the development team, regardless of their technical or coding expertise. Utilizing tools like Gherkin, automated test results, and automated test reports, BDD enables team members to accurately comprehend customer needs and continuously assess whether the software aligns with customer requirements in real-time. This collaborative approach ultimately facilitates the timely delivery of software products that meet customer expectations. For a comprehensive overview of the BDD software development process, please refer to Section 5.5, "Software development process of BDD."

Chapter 6

Conclusion

This chapter concludes the main task of this work and explains the achievements and the end state of the task.

6.1 Conclusion

The main task is to implement more functionality for ScreenTracer and ScreenTracerViewer to achieve more requirements of users. The main part of this work is the development phase, in which all mandatory requirements are implemented. Users can now store .avi format videos in specified directory through ScreenTracer and select video files from local directory through ScreenTracerViewer. Then, Users can see the Gherkin specification of the video at the same time, while the video is playing. According to the Gherkin specification, users can understand what is happening in the video and verify if the software product meets their requirements. Using ScreenTracer and ScreenTracerViewer in the software development process is beneficial for all team members involved. For the development team, ScreenTracer and ScreenTracerViewer are valuable tools for ensuring that software requirements are not overlooked and for preventing deviations from customer requirements that may occur due to interpretations of developers during the development process, and for the product manager and customer, ScreenTracer and ScreenTracerViewer offer a more convenient and intuitive way of verifying whether the product aligns with customer requirements. Implemented and how to implement requirements can be seen in chapter 4.

After the development phase, Test Cases were designed and planned to evaluate if the software meets all mandatory requirements. Because most of the requirements, tasks and achievements of this work are functional implementations.

6.2 Future Work

The current generation of test code using Selenium necessitates manual operation by developers. In future development phases, this process will be upgraded to be automated, while retaining the option for manual operation.

Appendix A

Appendix A

Contents of CD

Contents of the CD includes:

1. All source code under the branch "BA-new-driver"
2. Latex source code
3. The video of testing test cases
4. The bachelor thesis in digital form (.pdf)

Bibliography

[1] Mahmood Niazi, Sajjad Mahmood, Mohammad Alshayeb, Mohammed Rehan Riaz, Kanaan Faisal, Narciso Cerpa, Siffat Ullah Khan, and Ita Richardson. Challenges of project management in global software development: A client-vendor analysis. *Information and Software Technology*, 80:1–19, 2016.

[2] Ulrike Abelein and Barbara Paech. State of practice of user-developer communication in large-scale it projects: Results of an expert interview series. In *Requirements Engineering: Foundation for Software Quality: 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7-10, 2014. Proceedings 20*, pages 95–111. Springer, 2014.

[3] John Ferguson Smart and Jan Molak. *BDD in Action: Behavior-driven development for the whole software lifecycle*. Simon and Schuster, 2023.

[4] Jianwei Shi, Jonas Mönnich, Jil Klünder, and Kurt Schneider. Using gui test videos to obtain stakeholders' feedback. In *2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 35–45. IEEE, 2023.

[5] Mika V Mäntylä, Juha Itkonen, and Joonas Iivonen. Who tested my software? testing as an organizationally cross-cutting activity. *Software Quality Journal*, 20:145–172, 2012.

[6] Marlen Stacey Chawani, Jens Kaasbøll, and Sisse Finken. Stakeholder participation in the development of an electronic medical record system in malawi. In *Proceedings of the 13th Participatory Design Conference: Research Papers-Volume 1*, pages 71–80, 2014.

[7] Helge Holzmann. Videounterstützte ablaufverfolgung von tests für anwendungen mit grafischer benutzeroberfläche. In *Informatiktage*, pages 83–86, 2012.

[8] Raphael Pham, Helge Holzmann, Kurt Schneider, and Christian Brüggemann. Beyond plain video recording of gui tests: linking test case instructions with visual response documentation. In *2012 7th*

International Workshop on Automation of Software Test (AST), pages 103–109. IEEE, 2012.

[9] Christine Rupp et al. *Requirements-Engineering und-Management: Das Handbuch für Anforderungen in jeder Situation.* Carl Hanser Verlag GmbH Co KG, 2020.