# Software engineering final project

Project name: Quick Flattener

Authors: Adam Gembala, Robert Lotawiec

Degree course: Informatics, sem. V

Academic year: 2021/2022

Contents

[Software engineering final project 1](#_Toc95143308)

[Contents 2](#_Toc95143309)

[Document history 3](#_Toc95143310)

[Description 4](#_Toc95143311)

[Project's requirements 4](#_Toc95143312)

[Functional requirements 4](#_Toc95143313)

[Non functional requirements 5](#_Toc95143314)

[Project's dictionary 6](#_Toc95143315)

[Use case model 7](#_Toc95143316)

[Actors 7](#_Toc95143317)

[Use case 7](#_Toc95143318)

[Use case diagram 8](#_Toc95143319)

[Class diagram 9](#_Toc95143320)

[Package diagram 9](#_Toc95143321)

[Sequence diagram 10](#_Toc95143322)

[Test Plan 11](#_Toc95143323)

[Repository link 14](#_Toc95143324)

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Approved by** | **Additional information** |
| 07 Feb 2022 | Robert Lotawiec | Adam Gembala | Update of functional and non-functional requirement content |
| 07 Feb 2022 | Adam Gembala | Robert Lotawiec | Update of unit testing documentation |
| 07 Feb 2022 | Robert Lotawiec | Adam Gembala | Updating the content of acceptance, system, interface and application stress tests |
| 27 Jan 2022 | Robert Lotawiec | Adam Gembala | Adding a list of test proposals to the application |
| 26 Jan 2022 | Adam Gembala | Robert Lotawiec | Unit tests implementation |
| 25 Jan 2022 | Adam Gembala | Robert Lotawiec | Implementation of file mapping data structure |
| 04 Jan 2022 | Adam Gembala | Robert Lotawiec | GUI along with controller |
| 30 Dec 2021 | Adam Gembala | Robert Lotawiec | Changed project from UWP to WinForms, UI changes, implemented basic logging and strategy for algorithms |
| 9 Dec 2021 | Adam Gembala | Robert Lotawiec | Initialized UWP project with GUI prototype |
| 9 Dec 2021 | Adam Gembala | Robert Lotawiec | Updated package and class diagrams |
| 7 Dec 2021 | Adam Gembala | Robert Lotawiec | Added document’s history and table of contents |
| 5 Dec 2021 | Robert Lotawiec | Adam Gembala | Added sequence diagram |
| 2 Dec 2021 | Adam Gembala | Robert Lotawiec | Corrections of the use case model |
| 24 Nov 2021 | Adam Gembala | Robert Lotawiec | Updates to the package and class diagrams |
| 17 Nov 2021 | Adam Gembala | Robert Lotawiec | Updates to the class diagram |
| 16 Nov 2021 | Robert Lotawiec | Adam Gembala | Added initial package diagram |
| 13 Nov 2021 | Robert Lotawiec | Adam Gembala | Added initial class diagram |
| 8 Nov 2021 | Adam Gembala | Robert Lotawiec | Corrected use case diagram |
| 8 Nov 2021 | Adam Gembala | Robert Lotawiec | Updates to the project’s dictionary and requirements |
| 7 Nov 2021 | Robert Lotawiec | Adam Gembala | Updates to the project’s requirements |
| 7 Nov 2021 | Robert Lotawiec | Adam Gembala | Updates to the project’s dictionary |
| 3 Nov 2021 | Adam Gembala | Robert Lotawiec | Use case diagram and description of actors |
| 3 Nov 2021 | Adam Gembala | Robert Lotawiec | Corrections suggested by teacher |
| 3 Nov 2021 | Adam Gembala | Robert Lotawiec | Project’s functional and non-functional requirements |
| 2 Nov 2021 | Robert Lotawiec | Adam Gembala | Add files via upload |
| 2 Nov 2021 | Adam Gembala | Robert Lotawiec | Initial project dictionary |

# Document history

# Description

The goal of the project is to create an application with a graphical user interface for flattening nested directories with files. The user from the GUI can choose the source directory and the destination folder, where all the files from the nested source directory will be copied, the flattening operation must be reversible. After each operation, a log file will be created in which the history of the performed operations on files will be stored. In case of a conflict during copying files to the target folder (i.e., copied files have the same name), the user will be informed about it and will have to decide on his own about the occurred conflict (giving a new name to one of the files or skipping the copying of the file causing the conflict). The minimum requirement is that the application works with the Windows platform and that it works on desktop devices.

# Project's requirements

This section contains functional and non-functional requirements of the project.

# Functional requirements

* User should be able to interact with the program through a graphical user interface.
* User privileges should be checked at runtime and program should not stop its execution when user lacks the read or write privilege on some file or nested directory. In such case program should omit it and log such accident in log file.
* Minimal requirement is a Windows application. Windows and Linux compatibility is preferred.
* The program will consider all folders and files in the path and all file extensions.
* Program should allow user to resolve conflicts that may occur during execution of the algorithm. Conflict should stop algorithm until user will choose conflict resolution option in modal dialog.
* Program should log which file was mapped onto which name. Log file should also contain information about conflicts and the way they were resolved. Program should also log other accidents such as lacking privileges or the files that could not be opened (i.e. when file was opened in other application). Log file should have an extension '\*.log'.
* Operation of flattening is reversible through the GUI after providing the destination directory. Program should scan the directory if it's only one level deep and should itself find the log file by extension. In case of multiple log files user should be allowed to pick one and in case of nested directories program should prompt the warning but should allow to ignore them.
* During execution of reverse operation if program is not able to find a file that was specified in log, it should display the information about missing file, but should not stop the operation.

# Non functional requirements

* Application should display the information about conflicts between names of the files.
* User input should be tokenized.
* Conflict resolution dialog should allow user to specify a new name of the file, skip one or both.
* The program will work for every user of the operating system.
* The user must have permissions to the directory and its files in order to perform the flattening process. If the user does not have such rights, the program should display an appropriate message and continue the process only for folders and files to which he has rights in the source path.
* The files are copied to the new location without removing of the original files.
* The processing of each request should be done immediately.
* In order to undo the flattening process, the user specifies the directory of log file and selects the reverse algorithm, whereupon the program asks from the log file of which operation it should read the data of the flattening operation performed and perform the action of restoring the nested path. When the log file is not compatible with the selected algorithm the proper information will be presented, but user can still, despite of the warning, continue the execution with selected algorithm or chose auto-selection of algorithm, which is compatible with this log file.

# Project's dictionary

**Algorithm** - a set of instructions that makes a computer do the flattening process.

**Conflict** - In terms of flattening, the situation when two files are assigned with the same names.

**Conflict resolution** - Operation of resolving the conflict in file naming. Possible ways to resolve a conflict are skipping a file or rename it.

**Destination directory** - The directory to which files from the flattened directory will be saved.

**Input directory** - The directory that user specified to be flattened.

**Flattening process** - Flattening a directory means moving all the files from their various folder and/or subfolders into one parent folder in destination directory.

**Log file** - log file is a file that records events that occur after user's request for flattening process.

**Nested Directory** - A folder stored within another folder. A nested folder is also named subfolder.

**Notification** - Informing the user about conflict during program execution.

**Operating system** - The software that manages computer resources and creates the environment for running the program.

**Program** - Flattening algorithm written in a programming language.

**Search** - A process of searching the input directory to find all files to be flattened into one folder and moved to the destination directory.

**Subfolder** - Folder located inside Input/Destination directory hierarchy. Can contain other subfolders.

**User** - Person using the system for his or her own knowledge purposes.

**Reverse operation** - operation of reverting the algorithm execution.

# Use case model

# Actors

**User** - The regular user of the program. User wants to perform the operation of folder flattening. User is expected to have basic knowledge about program and file system. It is assumed that user has proper access levels to folders on which program will execute.

**Logger** - Service that performs write operation to '\*.flattening.log' file. It logs all events that may occur during program execution.

**File mapper** - Service that performs file mapping, detects conflicts and applies tokenization of user input.

**File system** - File system on which program operates.

# Use case

The program is expected to move all files from one folder hierarchy to another. Files in destination directory should have names that will allow to identify them. After operation of flattening the log file can be found inside destination directory. Log file contains all of the information about the files that were copied to destination directory. Example of the line from the log file looks like this:

|  |
| --- |
| source\_dir/subdir/file1.txt -> destination\_dir/tokenized\_name\_of\_file1.txt |

In case of conflict program displays the proper message and allows to amend the problem. Such situation produces record in log file:

|  |
| --- |
| [!] Conflict source\_dir/dir2/file1.txt -> destination\_dir/name\_of\_file1.txt  [+] source\_dir/subdir2/file1.txt -> destination\_dir/name\_of\_file1\_copy.txt |

# Use case diagram

Diagram

Description automatically generated

# Class diagram

Diagram

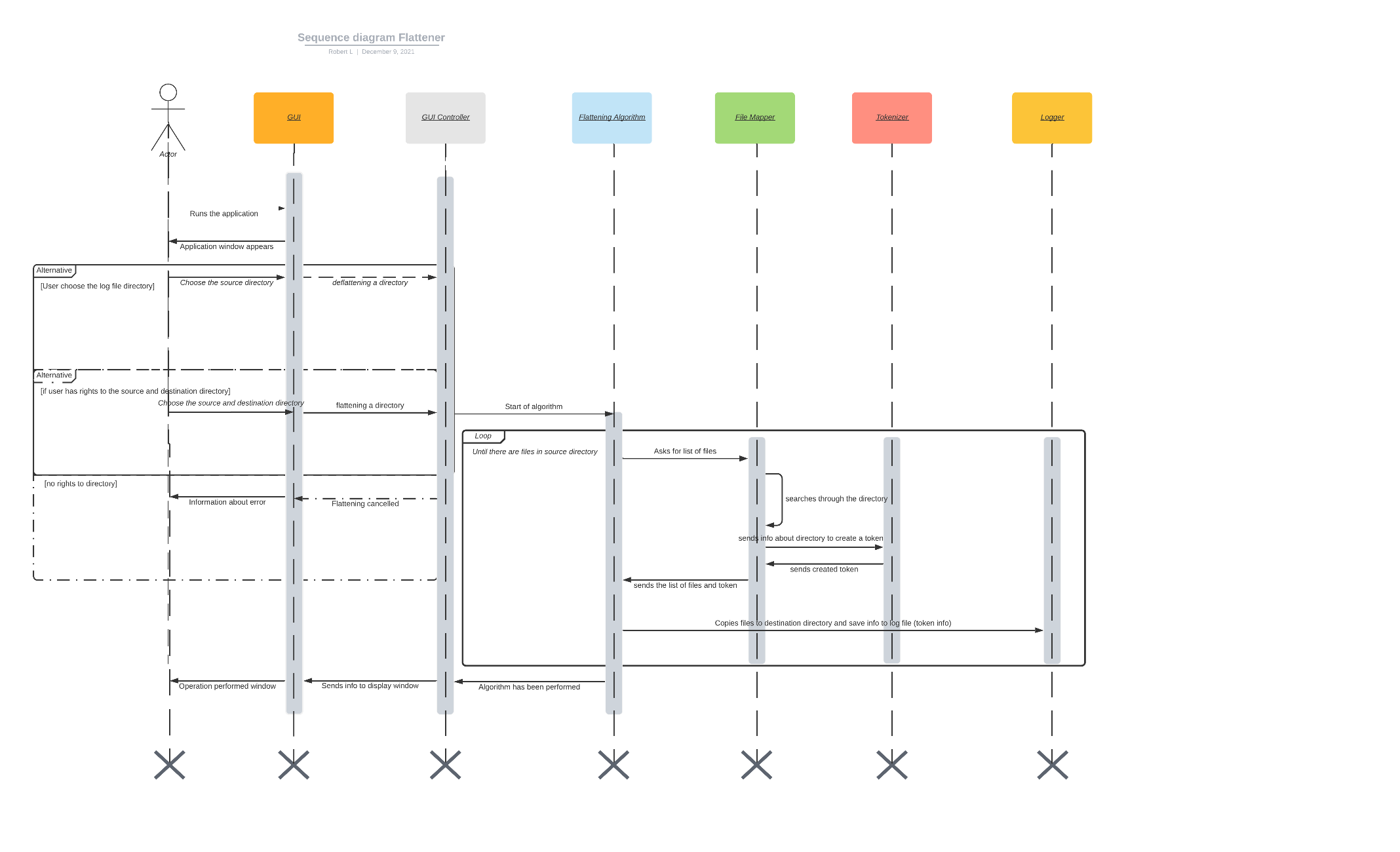
Description automatically generated

# Package diagram

Diagram

Description automatically generated

# Sequence diagram



# Test Plan

1. General description/ Scope of testing

The premise of our application testing plan is to focus on testing individual functionalities to ensure the best possible end-user experience and stability of the whole application. Unit, manual, interface and application stress tests have been scheduled to test these functionalities. Test categories, such as acceptance or system testing are designed to check functional and non-functional requirements.

1. Unit Testing

Unit tests are implemented with use of NUnit.

|  |  |
| --- | --- |
| Test case | Expected result |
| Tokenisation | |
| Test, if token values are changed to correct values. | Output of tokenisation should contain data specific to file. |
| Test, if characters that are not part of a token are rewritten to the output. | Sequence of characters that does not create a token should be omitted by tokenisation. |
| Data structure | |
| Data persistence test – checking if data written to the structure can be obtained in unchanged format. | Data that is written to the structure can be obtained unchanged at any moment |
| Data structure – conflict resolution | |
| Test if skipping file A is possible. | File A should not be present in mapped files after this operation. |
| Test if skipping file B is possible. | File B should not be present in mapped files after this operation. |
| Test if changing the name of the file A is possible. | File A should be present in mapped files with changed name. |
| Test if changing the name of the file B is possible. | File B should be present in mapped files with changed name. |
| Data structure – conflict detection | |
| Test if structure detects the proper number of conflicts. | Number of conflicts should be equal to the number of conflicting output files. |
| Test if structure detects the proper number of conflicting files. | Number of conflicting files should be equal to the number of files that are mapped to conflicting output. |
| File mapper | |
| Test if a single file is correctly mapped to the tokenised output. | Single file should be mapped to correct output without any conflict. |
| Test if two files mapped to the same output are detected as conflict. | Conflict should be visible in the number of total conflicts and in file mapping. |
| Test if CustomLogger invokes function assigned to it for each message. | Function passed to CustomLogger should be invoked for each message with unchanged massage as a parameter. |
| File parser | |
| Test if log file contents are correctly parsed to data structure. | Files and mappings from log file should be present in data structure. |

1. Acceptance testing

|  |  |
| --- | --- |
| Test case | Expected result |
| Test if program copied files to destination directory while keeping original files unchanged. | The files are copied to the new location. Original files are left untouched. |
| Test if program executes correctly as user and administartor. | Privilege level should be irrelevant for correct operation of the program. In case of lacking access program should log errors. |
| Test if user can select tokens in order to create output pattern. | User should be able to create output pattern without the necessity of knowing all token values. |
| Test if program omits localisation to which it has no access and reports such situations. | Program omits localisations to which it has no access and continues operation for remaining files. |
| Check that if a file in the location to be flattened is used by another application, it will be copied to the destination path. | File system verifies if there is no access conflict with another application to the given file at that time, in case of such a situation it throws an exception to log file and continues the process only for files to which there is no access conflict. |
| The user wants to perform a reverse operation and provides a log file that is not compatible with the selected flattening algorithm. | When the log file is not compatible with the selected algorithm the proper information will be presented, but user can still, despite of the warning, continue the execution with selected reverse algorithm or chose auto-selection of algorithm, which is compatible with this log file. |
| The user wants to perform a reverse operation and provides a valid log file to perform this operation, but its content is different from the current directory content. | User should be informed about missing files from the log file and continue the operation for other files. |
| The files to be copied to the destination directory, which come from different source subdirectories, have identical names. | Conflict stop algorithm until user will choose conflict resolution option in modal dialog. |

1. Application Stress Testing

|  |  |  |
| --- | --- | --- |
| Test description | Test case | Expected result |
| The goal of the test is to examine the performance of the application in case when the large amount of data is happen to be processed. | Huge number of files to process e.g., whole partition or 700 files each ~5MB. | Diplaying the message that program is still executing some tasks. Possibly give possibility to cancel currently running task. |
| “Big” files to process e.g. 20 files each ~2.5 GB. |

1. Interface testing

|  |  |
| --- | --- |
| Test case | Expected result |
| Test if user can easily change the flattening algorithm | User should be able to change algorithm in at most 3 clicks starting on main window |
| Test if user can easily create output pattern with use of tokens | User should be able to create output pattern without memorizing all of the tokens |
| Test if program disables possibility to start the algorithm in case of conflict or another invalid situation. | Program should disable actions that are related to launching of the algorithm in case of invalid situation and enable controls that allow user to assess the situation (e.g., conflict resolution button in case of conflict) |
| Test if program correctly displays information about conflicts. | Program should display the number of conflicts along with the number of conflicting files. |

1. System testing

|  |  |
| --- | --- |
| Test case | Expected result |
| Test if user can select flattening algorithm and if program provides a possibility of conflict resolution. | User can freely select one of available algorithms and manually resolve the conflicts. |
| Test if application creates the log file with information about mapped files and errors that occurred during processing of the input files. | Log file should be created in output directory. File should contain information about mapped files and (possibly) errors. |

1. Manual testing

|  |  |
| --- | --- |
| Test case | Expected result |
| Conflict resolution tests | |
| Test if in case of exception program logs the error to the file. | Exception message should be written to the log file with correct marking. |
| File system tests | |
| Test if program verifies if specified partition exists and if correctly logs error about missing partition. | Missing partition should be logged into log file with correct marking. |
| Test if program in case of unexisting output directory creates it and correctly logs such information. | Missing localisation should be created and a proper message should be written to the log file. |
| Logger tests | |
| Test if a log file is created if it does not yet exist | Log file should be created when the first message arrives. |
| Test if existing log file is not overwritten | New messages should be appended at the end of the existing file. |

# Repository link

<https://github.com/gemadam/SE-final-project>