Contents

Revision History

Date	Version	Description	Author(s)
10.10.2018	**1.0**	**First Draft**	**Ali, Chen, Liu**
13.10.2018	**2.0**	**Final Version (Finishing Part 2.1, 3.4, 3.6)**	**Ali, Chen Liu**

1 Introduction

1.1 Purpose

This is the Software Requirements Specification (SRS) document for ***AutoTasks***, a library written in programming language Eiffel, together with its 5 testing example programmes. This document describes scope of the library, both functional and non-functional requirements, design constraints, external interfaces, and other crucial aspects to provide a comprehensive overview of the library.

1.2 Scope

The project described in this document is a library written in Eiffel for tasks management. The code name of this project is AutoTasks. We will focus on several aspects of tasks managements:

- Create tasks and constraints
- Visualize all the tasks
- Produce topological sort of the tasks

The most useful benefits of AutoTasks are:

- Clear display of all the elements and constraints
- Quick and correct task orderings

1.3 Definitions, Acronyms and Abbreviations

Term/Abbreviation	Definition
Eiffel	Effiel is a type of staticall typed object-oriented programming language.
Library	A library is a collection of reusable code courses that provides modular functionalities to practical software developments.

Element	A vertex/node of a directed graph, representing an event in general (e.g. a task, a meeting).
Contraint	A constraint is an ordered element pair of form <element1, element2>, describing a relation between the two elements, typically modeled as a directed edge on a graph.
Cycle	In context of graph theory, a cycle is a path of vertices and edges wherein a vertex is reachable from itself.
Acyclic	The adjective “acyclic” is used to descibe a graph withou cycles.
Topological Order	Given a graph of elements and constraints, a sequence of all elements is said to be “in topological order” if for every directed edge u->v, u comes before v.
Topological Sort	the action of finding a valid element sequence that is in topological order
GUI	Graphical User Interface
CLI	Command-Line Interface
simple word	a sequence of ASCII characters not involving a space or new line

1.4 References

The following table list all related documents referenced in this SRS document.

No.	Reference & Document Title	Applicable Reference Version
1	Eiffel Style Guidelines	https://www.eiffel.org/doc/eiffel/Style_Guidelines
2	Topological Sort Page of Rosetta Code Site	https://rosettacode.org/wiki/Topological_sort
3	Definition & Introduction of Time Complexity	https://en.wikipedia.org/wiki/Time_complexity

1.5 Overview

Section 2 generally defines the functions, target users, constraints to be respected and the preliminaries required in order to define requirements. Namely, it provides further information into the specifications of the desired eiffel library as well as the 5 example programmes.

Section 3 lists the detailed specific functional and non-functional requirements of the library (and the 5 example programmes as well).

This requirement documentation was formatted and structured following the [IEEE 830-1998] standard.

2 Overall Description

2.1 Product Perspective

- AutoTask is a simple library with only logical subcomponents, as well as a GUI component for graph/cycle display.
- AutoTask does not involve any storage subcomponent or network subcomponent.

2.2 Product Functions

We present a general overview of all the functions that this library shall provide. A more detailed explanation of the functionalities is located in section 3.2.

2.2.1 Supported functions

The AutoTask library:

- allows user to enter string of elements what user wants to see in sequence.
- allows user to enter constraints of sorting.
- allows user to remove elements after taking input.
- allows user to remove constraints.
- will show a graphical representation of entered elements and constraints.
- will perform a topological sort.
- will document the cycle, if it exists, and then performing the topological sort of non-cyclic parts.

2.2.2 Unsupported functions

The AutoTask library will not:

- create a new element, if either element of a constraint has not been entered before.

2.3 User Characteristics

- The users of our AutoTask library are assumed to be Eiffel developers, who should be familiar with object oriented programming and the concept of “design by contract”.

- To use the library, the users should have necessary knowledge of topological sort as well.

2.4 Constraints

- In order to use AutoTask Library, each element of the graph must be uniquely identified by a “simple word” (see section 1.4 for definition) of length no greater than 100.

2.5 Assumptions and Dependencies

- The user are required to Windows, or a Linux/Unix (like) operating system.
- The user of the library should have a appropriate development environment of Eiffel installed. (e.g. EiffelStudio)
- To use the GUI provided by AutoTask library successfully, the user should install a graph visualization software “Graphviz”, and have EiffelVision library in place.
- To run the 5 example programmes of AutoTask, the use should have EiffelTime library in place.

3 Specific Requirements

3.1 External Interfaces

- To use the library during Eiffel development, the use need to call the library by passing parameters and received values the library returns thereafter.
- A GUI is only involved when displaying the current graph.

3.2 Functional Requirements

Notes for Priority: 1 (important), 2 (medium), 3 (less important)

Req. ID	3.2.01
Title	Enter Data\ Enter Element
Description	The library shall be able to take in a string as an identifier of new element, and create the new element accordingly.
Priority	1
Risk	C
Reference	None

Req. ID	3.2.02
Title	Enter Data\ Enter Constraint
Description	The library shall be able to take in two strings as identifiers for the two elements of a new constraint, and create the new constraint accordingly.
Priority	1
Risk	C
Reference	None

Req. ID	3.2.03
Title	Enter Data\ Enter Element\ Lifetime of Data
Description	The library will keep all entered elements (cf. 3.2.01, 3.2.02) and constraints until removed (cf 3.2.04, 3.2.05) or the programme execution is terminated.

Priority	1
Risk	C
Reference	3.2.01, 3.2.02, 3.2.04

Req. ID	3.2.04
Title	Remove Data\ Remove Element
Description	The user should be able to remove an element from the library by specifying a string as the identifier.
Priority	1
Risk	H
Reference	None

Req. ID	3.2.05
Title	Remove Data\ Remove Constraint
Description	The user should be able to remove a constraint from the library by specifying 2 strings as the identifier.
Priority	1
Risk	H
Reference	None

Req. ID	3.2.06
Title	Graph Processing\ Cycle Detection
Description	Given elements and constraints, the system shall be able to detect a cycle and if it exists, return all elements and constraints of that cycle.
Priority	2
Risk	M
Reference	3.2.03

Req. ID	3.2.07
Title	Graph Processing\ Topological Sort\ No Cycle

Description	The library shall be able to perform topological sort if no cycle is detected (cf. 3.2.06).
Priority	1
Risk	M
Reference	3.2.06

Req. ID	3.2.08
Title	Graph Processing\ Topological Sort\ With Cycle
Description	If a cycle is detected (cf. 3.2.06), the library shall be able to perform topological sort on the non-cyclic parts for the graph
Priority	2
Risk	M
Reference	3.2.06

Req. ID	3.2.09
Title	Display\ Display Graph
Description	The library shall be able to display graphical representation of entered elements (cf. 3.2.01) and constraints (cf 3.2.02) using the visualisation software Graphviz.
Priority	2
Risk	H
Reference	3.2.01, 3.2.02, 3.2.03

Req. ID	3.2.10
Title	Display\ Display Cycle
Description	The library shall be able to display the graph of cyclic parts if detected (cf. 3.2.06).
Priority	2
Risk	H
Reference	3.2.06

Req. ID	3.2.11
Title	Input Integrity\ Element\ Duplication
Description	If the newly entered element (cf. 3.2.01, 3.2.04) has the same name as one of the existing elements (cf. 3.2.03), the library shall be able to omit the newly entered one and return warning messages.
Priority	2
Risk	L
Reference	3.2.01, 3.2.03, 3.2.04

Req. ID	3.2.12
Title	Input Integrity\ Element\ Exceed Max Length
Description	If the newly entered element (cf. 3.2.01, 3.2.04) has a name longer than 100 characters, the library shall be able to omit the newly entered element and return warning messages.
Priority	2
Risk	L
Reference	3.2.01, 3.2.04

Req. ID	3.2.13
Title	Input Integrity\ Constraint\ Undefined Element
Description	If the newly entered constraint (cf. 3.2.02, 3.2.05) specifies one or two non-existing elements, the library shall be able to omit the newly entered constraint and return warning messages.
Priority	2
Risk	L
Reference	3.2.02, 3.2.05

Req. ID	3.2.14
Title	Input Integrity\ Constraint\ Exceed Max Length
Description	If the newly entered constraint (cf. 3.2.02, 3.2.05) contains one or two element names longer than 100 characters, the library shall be able to omit the newly entered constraint and return warning messages.

Priority	2
Risk	L
Reference	3.2.02, 3.2.05

Req. ID	3.2.15
Title	Example Programm\ 1st example
Description	The first example programme should take in inputs from console in a format that is specified by “Topological Sort Page of Rosetta Code Site” (see section 1.4), and then perform topological sort accordingly.
Priority	2
Risk	L
Reference	None

Req. ID	3.2.16
Title	Example Programm\ 2nd example
Description	The second example programme should take in a simple make file consisting of lines of strings. Each line is in the following format: elem_0 : elem_1 elem_2 ... elem_n The output should be a list of all the elements appearing in the input file, in an order such that, if there is a line such as the above, elem_1, elem_2, ... elem_n appear before elem_0.
Priority	2
Risk	L
Reference	None

Req. ID	3.2.17
Title	Example Programm\ 3rd example
Description	The third example programme should take in a simple make file with the same format specified in the 2nd example programme (cf. 3.2.16). The third example should contains 4 different elements, 10 constraints, and one or more cycles.

Priority	2
Risk	L
Reference	3.2.16

Req. ID	3.2.18
Title	Example Programm\ 4th example
Description	The fourth example programme should take in a simple make file with the same format specified in the 2nd example programme (cf. 3.2.16). The third example should contains 200 different elements, and 1000 constraints, with no cycles.
Priority	2
Risk	L
Reference	3.2.16

Req. ID	3.2.19
Title	Example Programm\ 5th example
Description	The fifth example programme should take in a simple make file with the same format specified in the 2nd example programme (cf. 3.2.16). The third example should contains 2000 different elements, and 100,000 constraints, with no cycles.
Priority	2
Risk	L
Reference	3.2.16

3.3 Non-Functional Requirements

Req. ID	3.3.01
Title	Availability
Description	The library shall be available to perform whole functionality 24/7.
Priority	1
Risk	C

Reference	3.2.03

Req. ID	3.3.02
Title	Reliability
Description	The library shall always work correctly without any corruption.
Priority	1
Risk	C
Reference	3.2.07, 3.2.08

Req. ID	3.3.03
Title	Recoverability
Description	The library shall be able to take input again after the wrong input has been entered.
Priority	1
Risk	C
Reference	3.2.11, 3.2.12, 3.2.13, 3.2.14

3.4 Performance Requirements

Req. ID	3.4.01
Title	Performance of input
Description	The library shall be able to process input data in not more than 1000 milliseconds.
Priority	2
Risk	L
Reference	3.2.01, 3.2.02

Req. ID	3.4.02
Title	Performance of Graph Display
Description	The library shall be able to draw graph in not more than 500 milliseconds.

Priority	2
Risk	L
Reference	3.2.09

Req. ID	3.4.03
Title	Time Complexity (see section 1.4 for introduction of Time Complexity)
Description	The time complexity of adding an element would be O(1) (cf. 3.2.01). The time complexity of adding a constraint would be O(1) (cf. 3.2.02). To delete an element ***a*** (cf. 3.2.04), the time complexity would be O(C) , where C is the number of constraints related to element ***a***. To delete a constraint (cf. 3.2.05) from element ***a*** to another element ***b***, the time complexity would be O(C), where C is the number of constraints that are from element ***a***. Suppose N elements and M constraints, the library should perform cycle detection (3.2.07) or topological sort with a time complexity of O(N + M) (cf. 3.2.08).
Priority	1
Risk	L
Reference	3.2.01, 3.2.02, 3.2.04, 3.2.05, 3.2.06, 3.2.07, 3.2.08

3.5 Maintainability

To enhance the maintainability, the source codes of the library should follow these rules:

- The source codes should be appropriately indented, and 4 white-spaces should be used at each level of indentation.
- Variables should be used in a manner by the Effile Style Guidelines (section 1.4)
- Notes / comments should be used for each class to identify the functionalities and design consideration if applicable.
- Git and GitHub should be used during the whole development process, so as to achieve better version controlling.

3.6 Design Constraints

- The library should be built with Eiffel as the progrgamming language.
- The IDE for development of the Library should be EiffelStudio.
- The software “Graphviz” would be used for visualisation of the graph.
- The library “EiffelVision” and “EiffelTime” would be used.