University of Zurich^UZH

Software Requirement Specification for

<YODA>
YOUR OUTRIGHT DOCUMENT ASSEMBLER

Daniela Flüeli
Joel Barmettler
Marius Högger
Spasen Trendafilov

Supervision:
Prof. Bertrand Meyer

Software Engineering – Department of Informatics
University of Zurich

October 15, 2017 | Zurich, Switzerland

Table of Contents

List of Figures

List of Tables

Revision History

TABLE 1: LIST OF ALL THE MEETINGS AND REVISION OF THE YODA TEAM

Date 2017	Description	Author(s)
27.9	Kickoff Meeting, Problem identification	Team-Meeting
29.9	UML Class Diagram Prototype	Team-Meeting
1.10	Requirement Analyzation, Description Notes	Team-Meeting
4.10	Diagrams, Descriptions and Tables, References	Team-Meeting
8.10	Reviewing SRS, Dealing with Inconsistency	Team-Meeting
13.10	Reviewing SRS, Final Touches	Team-Meeting

1 Introduction

This Introduction provides an overview of the entire Software Requirements Specification (SRS) for the Markup generator *YODA (Your Outright Document Assembler)*. First, we specify the purpose of this SRS, highlighting the importance of a complete and comprehensive SRS in terms of saving time and money. Thereafter we provide a brief description of the scope as well as the boundaries of *YODA*. The next two subsection represents on the one hand a table with definitions, acronyms and abbreviations needed to properly interpret the SRS and on the other hand a complete list of all documents and websites consulted during creating this SRS. The last part of this introduction gives an overview of the further content, structure and organization of the SRS.

1.1 Purpose

Good specifications are essential for a software project to be successful in regard of task execution, teamwork and costs. Since we have the mandate to build a Markup generator named *YODA*, we wrote this SRS. There are several different stakeholders with different needs involved in the development of our software project. The process of writing a SRS connects these different stakeholders like customers and engineers to establish a common understanding of what *YODA* should be able to do. With respect to implementation the SRS facilitates the division of labour by providing a reference work. All involved engineers are obliged to consult the SRS to ensure doing the right thing. The SRS may serve to justify towards our stakeholders that the defined goals are met. Explicitly quantified requirements adhering the quality goals of verifiability and traceability help to verify the correctness of the implementation and increase the probability of being achieved. This SRS gives a complete and comprehensive description of *YODA* including (non-)functional requirements and design constraints. It defines precisely what we are going to build in order to save time, costs and prevent from dissatisfied stakeholders that leads to damage of our image.

1.2 Scope

This section includes a brief description of the scope as well as the boundaries of *YODA*, both described in more detail in section 2.3 Product Functions.

YODA is a Markup-Content generator library, written in Eiffel. *YODA* relieves the client from the burden of generating syntactically correct Markup-Code from data all on his own. The interaction between the actor and *YODA* is visualized in the following use case diagram and described below.

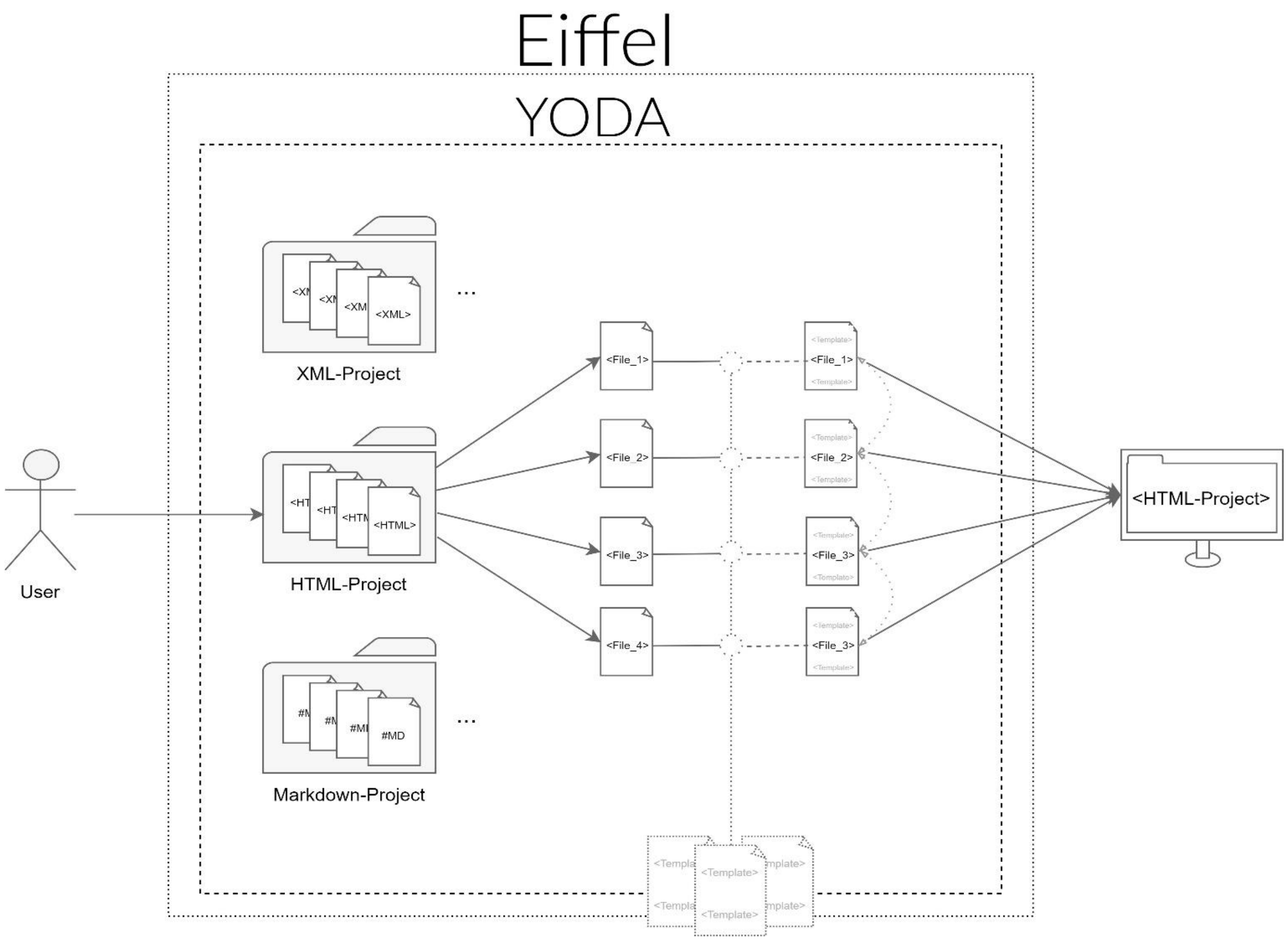


FIGURE 1: USE CASE MODEL OF THE YODA LIBRARY

First, a programmer imports *YODA* to its development environment. Now the client can facilely generate documents in Markup format by using *YODA*. For this purpose, the client instantiates a *YODA*-Project of a specific Markup-language such as HTML, XML or Markdown and chooses a corresponding file template with predefined placeholder-tags marking where generated content will be inserted. Afterwards he can add several *YODA*-Files of the same language to this project with the order corresponding to the order in the Output-Document. To add elements as for instance text, lists, tables or images to the files the client is advised to use *YODA's* wrapper-functions to spare himself creating instances of *YODA*-Elements manually, even though this is a practicable approach, too. The wrapper-functions may be concatenated into each other in order to create different concatenation levels corresponding to the nested structure of Markup-languages. Code snippets already written in a Markup-language can be easily included in the document, too. However, *YODA* does not do any kind of correctness-check to these imported code snippets, instead the client has to ensure that they are of the same Markup-language as the *YODA*-Project type as well as that they are syntactically correct. In addition *YODA* provides simple styling of text elements such as making part of the text bold, italic or underline. Furthermore, *YODA* comes with methods to print out all names of *YODA*-Files contained in a *YODA*-Project as well as all names of *YODA*-Elements contained in a *YODA*-File to the console as a matter of survey. *YODA* offers the client for some Markup-Languages moreover the ability to add links to *YODA*-Files, both external to URLs in the world wide web and internal to other *YODA*-Files in the same *YODA*-Project.

However, at time of first release *YODA* supports the generation of HTML documents only. But its architecture allows high extensibility with respect to output documents written in other Markup-languages as XML or Markdown.

1.3 Definitions, Acronyms and Abbreviations

To properly understand the terminology of this SRS, this table shows all titles with the matching descriptions. The reference numbers WX (Websource + and DX) correspond to the following references in the next table later on.

TABLE 2: DEFINITIONS OF ALL USED TERMS THAT DO NOT BELONG TO COMMON KNOWLEDGE

#References	Title	Description
W12	Academics	Highly educated person in their studies or a graduate of a university or college.
W17	Adaptive Maintenance	An aspect of maintainability, describing the facility of adapting a system prompt and easily to changes in the environment.
	Administrator Rights	Includes the power of decision over the incorporation of commits 1. the composition of the *YODA*-Administrators 2. the declaration of reported bugs as proven and list them in the according list on the *YODA*-Bug-Board.
D1	Class	All Eiffel code must exist within the context of a *class*
W18	Class Diagram	UML class diagram that shows the programmatic structure
	Client	Stakeholder that in some way directly interact with YODA.
	Console	Terminal which allows the Developer to Input data into a software and directly receive output.
	Container	Abstract object that contains a certain amount of specific data.
W17	Corrective Maintenance	An aspect of maintainability, describing the facility of bug detection and bug fixing, in case of a malfunction or breakdown of the system.
D3	correctness	Correctness is defined as the software's ability to perform according to its specification
W8	Eiffel	Object oriented programming language
W13	EiffelStudio	Integrated Development Environment for Eiffel
	Encapsulation	Enclose or be enclosed in or as if in a capsule
W22	GitHub	Web Based File Hosting-Service used for sharing Software Code
W11	GNU License 2.0	License out of the GNU Library General public license from 1991.
W1	HTML	Hypertext Markup Language used for building Websites
W19	HTML-Footer	Part of a HTML-Document that is located at the very bottom and is shared between multiple files. It contains legal information like Contact information, imprint or policies.
W20	HTML-Head/Header	Part of a HTML-Document that is located at the very top and is shared between multiple files. It contains HTML-Metadata, links and descriptions of the files and projects.
W21	HTML-Metadata	Invisible information in a HTML document that is written inside the header and addresses robots like Search-Engine crawlers.
W27	Instance	An instance of a Class is an object, following the architecture defined in the class.
W28	Interface-Segregation	Large interfaces should be split into smaller and more specific ones so that clients will only have to know about the methods that are of

	Principle	interest to them.
W9	Library	A software library is a suite of data and programming code that is used to develop software programs and applications. It is designed to assist both the programmer and the programming language compiler in building and executing software.
W29	Maintainability	A design consideration concerning the ease with which *YODA* can be maintained once it is released and running.
W5	Markdown	A lightweight Markup language with plain text formatting syntax
W23	Markup	A File written in a language following certain rules and using certain keywords that specify behaviour and layouts of the text.
W23	Markup-language	A language that annotates text so that the computer can manipulate that text
	Menu	Section of the HTML-Page that links to all available Pages and Subpages in a Project.
	Nesting (Layer)	Nesting describes a recursive structure where objects contain other similar objects. The number of nesting layers describe how many such objects are encapsulated in each other.
UML Diagram	non-terminating	Continuous a Process, requires further encapsulation.
W10	Open Source Software	Software that follows certain defined criteria like having accessible source code and allowing giving away software parts through 3. Parties.
	Open-Closed Principle	Modules should be open and closed.
	Output type	The file-type that corresponds to the chosen Markup-language, like ".html".
	Parsing	Checking input on context correctness and possible causes of failure.
W24	Perfective Maintenance	An aspect of maintainability, describing the extendibility of a system in order to respond to changed stakeholder requirements, both in terms of function and efficiency.
W6	Placeholder Tag	Special Marker in the template that will be recognized and replaced by generated Content.
W24	Preventative Maintenance	An aspect of maintainability, describing the facility of anticipating risks as well as the understandability of a system achievable through consistent coding style and comprehensive documentation.
	Print (to Console)	Commanding the program to output certain information to the console to make it readable for the user.
W25	RAM	Random Access Memory, volatile data storages used in computers
D3	reliability	general term for correctness and robustness
	Rendering	Process of forming abstract data into a visible, usable result.
D3	robustness	Robustness is defined as its ability to react to cases not included in the specification
D3	Single Responsibility Principle	Every module has responsibility over a single part of the functionality provided by the software, so that it has only one reason to change.
D3	Single-Choice-Principle	Whenever a software system must support a set of alternatives, one and only one module in the system should know their exhaustive list.
W3	Snippets	An independent, self-contained piece of text in a Markup- or programming language
	Software Project	A general Project that a Programmer is working on and tries to fulfil.
W16	StackOverflow	Website to learn, share and improve code
W7	Stakeholder	All the people that in any way deal with YODA
W14	Subpage	Page that is listed under another Page in the *HTML*-Site Menu.

W6	Template	Pre-layouted file with pre-existing content and specific Placeholders.
UML Diagram	terminating	Finishing a Process, terminating encapsulation.
D3	Uniform Access principle	Facilities managed by a module are accessible to its clients in the same way whether implemented by computation or by storage.
	Wrapper-function	Function around a *YODA*-Object that autonomously creates and places objects without forcing the user to deal with instances and creations.
W26	XML	Extensible Markup Language used for structure and format data and information
Name	*YODA*	Markup generator library
	YODA-Administrators	A subset of the *YODA*-Community, including testers, managers and the support team with administrator rights and maintenance responsibilities.
	YODA-Bug-Board	A board on the *YODA*-GitHub-Repository with the objective of bug detecting and communicating. It consists of two lists, one contains supposed bugs reported by the *YODA*-Community, the other bugs proven by the *YODA*-Administrators.
	YODA-Community	The collectivity of all *YODA*-Programmers.
	YODA-Documentation	Readable Text File that ships with *YODA* and explains all the functionalities and usages of *YODA* for the identified stakeholders.
UML Diagram	*YODA*-Element	An instance of the Element Class, which is part of the *YODA*-Library.
UML Diagram	*YODA*-File	An instance of the File Class, which is part of the *YODA*-Library.
	YODA-Main-Functionality	Guarantees the ability to generate documents in at least one Markup format consisting of at least the most important *YODA*-Elements as well as the fundamentals of working with *YODA*-Projects, *YODA*-Files and *YODA*-Elements.
	YODA-Methods	All methods provided by *YODA.*
	YODA-Programmers	The Programmers working on the open source code of *YODA*. Forming in a body the *YODA*-Community.
UML Diagram	*YODA*-Project	An instance of the Project Class, which is part of the *YODA*-Library.
	YODA-Requirements-Checklist	Table to facilitate keeping congruence between source code and requirements. One row for each requirement and one column for every update.

1.4 References

This Subsection provides a complete list of all documents and websites that we used for this SRS.

1.4.2 Documents

TABLE 3: LIST OF ALL DOCUMENTS USED AS REFERENCES IN THE DEFINITIONS

#Doc	Title
D1	110_softcons_intro.pdf – Bertrand Meyer
D2	ex_week2_final.pdf – Tutorial about Git and Eiffel
D3	Object-Oriented Software Construction, second edition – Bertrand Meyer 1988

1.4.2 Websites

TABLE 4: LIST OF ALL WEBSITES USED AS REFERENCES IN THE DEFINITIONS

#Web	Title
W1	https://de.wikipedia.org/wiki/Hypertext_Markup_Language

W2	https://www.w3schools.com/css/css_intro.asp
W3	https://www.gruenderszene.de/lexikon/begriffe/snippet
W4	https://www.eiffel.org/doc/eiffel/ET%3A%20Inheritance
W5	https://en.wikipedia.org/wiki/Markdown
W6	https://de.wikipedia.org/wiki/Webtemplate
W7	http://wirtschaftslexikon.gabler.de/Definition/anspruchsgruppen.html
W8	http://www.eiffel.org
W9	https://www.techopedia.com/definition/3828/software-library
W10	https://opensource.org/osd
W11	https://opensource.org/licenses/LGPL-2.0
W12	http://www.macmillandictionary.com/dictionary/british/academic_1
W13	https://www.eiffel.com/eiffelstudio/
W14	https://www.w3schools.com/html/
W15	http://www.dictionary.com/browse/encapsulate
W16	https://stackoverflow.com/company
W17	http://www.businessdictionary.com/article/599/corrective-vs-adaptive-maintenance-for-your-business/
W18	http://study.com/academy/lesson/what-is-a-uml-class-diagram-definition-symbols-examples.html
W19	https://www.w3schools.com/tags/tag_footer.asp
W20	https://www.w3schools.com/tags/tag_header.asp
W21	https://www.w3schools.com/tags/tag_meta.asp
W22	https://github.com/features#code-review
W23	https://techterms.com/definition/markup_language
W24	http://searchitoperations.techtarget.com/definition/preventive-maintenance
W25	https://www.computerlexikon.com/was-ist-ram
W26	https://wiki.selfhtml.org/wiki/XML
W27	https://www.codecademy.com/en/forum_questions/558cd3fc76b8fe06280002ce
W28	http://www.oodesign.com/interface-segregation-principle.html
W29	http://www.businessdictionary.com/definition/software-maintainability.html

1.5 Overview

The further content, structure and organization of our SRS is abstracted in this section.

1.5.1 Section 2. Overall Description

focuses on general factors affecting *YODA* which have to be incorporated during specifying the requirements.

First, insight into already existing Markup-Content generators is delivered. Followed by a description of the product perspective. The section Product Functions gives an overview of supported and unsupported functions. In addition, it provides a differentiation between shared and specific features of the tree Markup-Languages taken into account. Then a short section deals with an accurately definition of the YODA-Client. The next section is about constraints for example due to Interfaces to other applications, hardware limitations, regulatory policies or environmental limitations. Finally, some assumptions which underlie the requirements are mentioned as well as some dependencies.

1.5.2 Section 3. Specific Requirements

lists all software requirements defined for *YODA* as much consistent, complete, precise and concise as possible.

Functional requirements are listed first, partitioned into requirements related to *YODA*-Projects, *YODA*-Files and *YODA*-Elements. Followed by requirements referring to the usability and reliability of *YODA*. Performance Requirements as specific response times or resource limitations are described next. Afterwards requirements enhancing the maintainability of *YODA* are specified. In the end, requirements concerning design constraints due to mandated design decisions as the used programming language are listed.

1.5.3 Section *4.* Supporting Information

At the very end, an index and an appendix are provided as supporting information in order to make the SRS easier to use. The index lists all requirements by its ID, providing as further information the title as well as the page on which the requirement can be found. The appendix contains a first sketch of a class diagram, we draw to get a better idea of how YODA might look like. Additionally, a summarization of the naming and comment convention described in "Object-Oriented Software Construction" is provided to serve the *YODA*-Programmers as an overview, but does not replace the reference book "Object Oriented Software Construction".

2 Overall Description

The following chapter presents an overall description of the *YODA* including current solution, product functions, user characteristics, constraints, assumptions and dependencies. This section does not contain any specific requirements it is meant to outline the background for those requirements.

2.1 Current Solution

HTML format and other Markup-languages make data better readable and thus find many applications in software projects. There are multiple ways to include HTML formats into software project. First of all, there's the possibility to include raw HTML in string objects, which means that the HTML gets handled as normal strings, this however leads to messy code and is very inconvenient to work with. Also, there's the way to exporting the data and then creating an HTML file using any external program or web based service. More convenient is to use a library which handles all the HTML syntax but hold on to the HTML functionality with its nesting ability and styling options. There are already some HTML generating libraries available on the internet and also new ones will be written surely, however with *YODA* we want to solve the problem in our very own way.

2.2 Product Perspective

YODA is a library and thus does not include any kind of Graphical User Interface (GUI). When included into an software-project, *YODA* enables functionalities to wrap data from the current software-project into a well readable format such as HTML, XML or Markdown without leaving the Eiffel-editor. This allows for simultaneous data processing and data formatting within the editor and the exporting of complete Markup-Files to local repositories.

Since the interface to the library is the Eiffel Programming language, some Eiffel programming knowledge is required to use *YODA*.

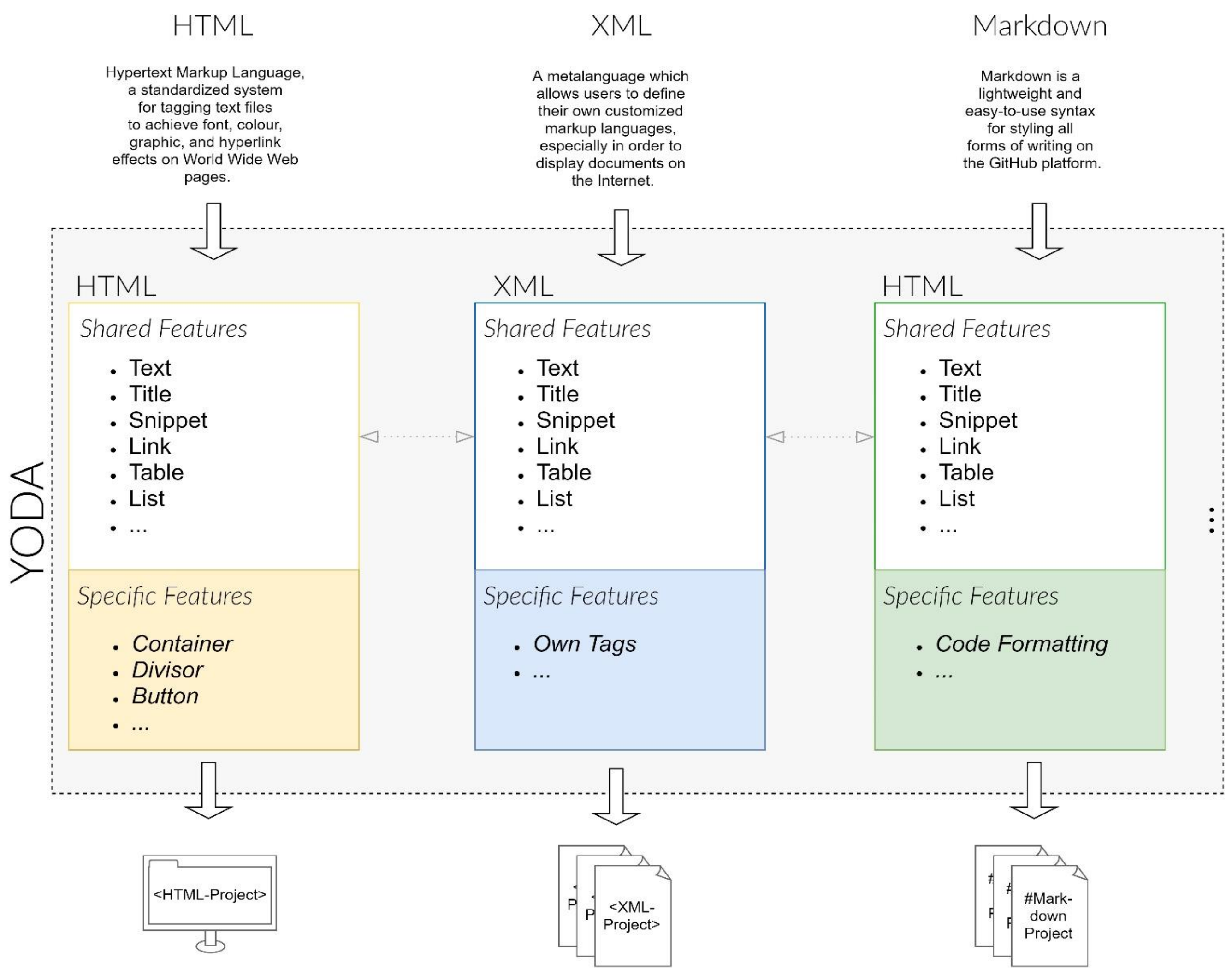
YODA is intended to support the client in formatting data into Markup-format and will generate the markup-code. However, the client shall provide some knowledge on nesting structures or read the *YODA*-Documentation carefully in order to use the whole potential of *YODA*.

To generate the files *YODA* stores the used data-files such as the template, images, snippets and more, on the local drive. Hence the memory consumption includes all these used files and additionally some Kilobytes for the finished Markup-file. The memory management is not part of *YODA* and is incumbent upon the client.

YODA is extensible towards other Markup-languages and new functionalities within the Markup-language.

How *YODA* is used is not part of this SRS but will be topic of the *YODA*-Documentation.

2.3 Product Functions



HTML
Hypertext Markup Language, a standardized system for tagging text files to achieve font, colour, graphic, and hyperlink effects on World Wide Web pages.
XML
A metalanguage which allows users to define their own customized markup languages, especially in order to display documents on the Internet.
Markdown
Markdown is a lightweight and easy-to-use syntax for styling all forms of writing on the GitHub platform.
YODA
HTML
Shared Features
Text
Title
Snippet
Link
Table
List
...
Specific Features
Container
Divisor
Button
...
XML
Shared Features
Text
Title
Snippet
Link
Table
List
...
Specific Features
Own Tags
...
HTML
Shared Features
Text
Title
Snippet
Link
Table
List
...
Specific Features
Code Formatting
...
<HTML-Project>
<XML-Project>
#Mark-down Project

FIGURE 2: SHARED FEATURES OF YODA BETWEEN THE SUPPORTED MARKUP LANGUAGES

The model above provides a proper sense of the Shared Features and the Specific Features of the planned three Markup-languages. Shared Features remain the same across the different languages. Specific Features on the other hand separate the different Markup-languages from each other. For each type of project, a certain output is generated through *YODA*.

2.3.1 Supported Functions

YODA does provide

- creating Output for a variety of Markup-languages
- creating a *YODA*-Project of one of the supported Markup-language
- creating *YODA*-Files and adding them to *YODA*-Projects.
- creating *YODA*-Elements and adding them to *YODA*-Files
- nesting *YODA*-Elements, if supported by the Markup-language
- easy to use function for creating and nesting *YODA*-Elements (wrapper-functions)
- individual *YODA*-Elements for the basic Markup-language-Elements
- Templates in which the created content gets injected
- preview of current content converted to the corresponding Markup-language
- rendering a project with all its files and elements into a Template to a local folder.
- support of HTML as a Markup-language
- Templates for HTML

2.3.2 Unsupported Functions

YODA does **not** provide

- any kind of version control for the client,
- code management or code storage,
- parsing,
- the editing the arguments of elements after instantiation,
- saving and exporting of elements outside of the software project,
- any kind of correctness-check on imported code snippets,
- *YODA* projects with different output files

2.4 User Characteristics

YODA is designed for clients with sophisticated expertise in software development and particularly with software projects. Hence the client shall provide a basic level of programming experience in Eiffel such as using libraries, converting data types as well as handling objects and features. People with no technical understanding and no experience in the use of computers are not the category of people *YODA* is designed for.

2.5 Constraints

This SRS represents the state of development before taking any Design Patterns into account. Consequently this SRS is formulated in a rather general form.

Interfaces to other applications

- *YODA* requires the installation of Eiffel.

High order language limitation

- *YODA* is entirely written in Eiffel 17.05 and can maybe not be fully used in older Eiffel Versions as well as other languages such as C++, Python or Java. *YODA*'s output is constrained by the general output format of the supported Markup-languages.

Hardware limitation

- *YODA* is constrained by the fact that it relies on the client to provide the needed read and write permissions to the client's hard disk. In addition, a shortage of the client's RAM and disc space can limit *YODA* in its capabilities.

Reliability requirement

- *YODA's* templates are accessible for editing and modifying, however *YODA* is not intended to assert the correctness of modified templates. This lies in the responsibilities of the client, same applies for snippets
- The speed in which *YODA* does operate is highly affected by the amount of data the client wants to use.

Regulatory policies

- *YODA* is intended to be released as open source to the public, therefore it must fulfill the respective licensing requirement.

Parallel operation

- *YODA* allows threading, as long as the individual threads don't conflict with each other, namely claim access to the same variables and files, which can happen while rendering the output files.
- The client needs to make sure all the files and content *YODA* requires for rendering the project are in a static folder on the local disk and are not in current use of any other program.

Environmental limitation

- This document as well as *YODA* itself are part of a one-semester-project, hence time acts as a limiting factor.

Duration of maintenance

- The current YODA-Administrators take the liberty of limiting the guaranteed maintenance to a year after the first release of *YODA*. All in section 3.6 defined requirements are restricted by this constraint.

2.6 Assumptions and dependencies

YODA's workability highly depends on correct and complete installation and integration into the software project such that all of *YODA*'s source files are stored locally and are accessible by the editor.
YODA's workability depend on the backwards compatibility of future updates of Eiffel. Consequently, it is assumed that the client's operating system supports the use of Eiffel
It is assumed that the general structure and syntax of the supported output types does not change in a way that makes *YODA*'s output unusable over the next 5 years. This assumption is based on the changes over the last 5 years.
All statements made in this SRS are made under the assumption that *YODA*'s source files are not directly modified by the client.
It is assumed that the client possesses a basic knowledge of English.

3 Specific Requirements

3.1 Functional Requirements

Requirement ID | Title	**ID**: x.x.x.xx, Unique identifier for each requirement within the SRS document that serves as a group indicator. **Title:** Individual, meaningful and descriptive name for each requirement, defines group to which the requirement belongs
Reference	Shows relation to other requirements that are relevant in this context.
Description	The definition of the requirement.
Priority | Risk	**Priority:** Defines in which order the listed requirements should be implemented. Priority ranges from 1 to 3 with 1 being mandatory requirements for the first implementation, 2 being mandatory for the final submission and 3 being optional to implement at all. **Risk:** States how critical the effect of not implementing the requirement is for the System in order to work correctly and deliver the expected output. The following Risk-Levels are defined: ● S (*Small Risk*): The validation of the requirement will only have a local effect and does not constrain the *YODA*-Main-Functionality. ● M (*Medium Risk*): The violation of the requirement will cause side effect to other requirements, but will not constrain the *YODA*-Main-Functionality. ● L (*Large Risk*): The violation of the requirement will constrain the *YODA*-Main-Functionality partially without an imminent risk of a breakdown of YODA.

	● XL (*Extra Large Risk*): The violation of the requirement will constrain the *YODA*-Main-Functionality profoundly with an imminent risk of a breakdown of *YODA*.

3.1.1 Project Related Requirements

ID | Title	1.1.1.1 | **YODA-Project, Container of Files and attributes**
Reference	R. 1.2.1.1, R. 1.1.6.1
Description	The client shall be able to create *YODA*-Projects that serve as a Container of related *YODA*-Files and project attributes. Each *YODA*-Project shall have a client-chosen name as an attribute as well as a valid link to a valid, type-matching template.
Priority | Risk	1 | XL

ID | Title	1.1.2.1 | **YODA-Project Type, Project-Output relation**
Reference	-
Description	For each supported output type, there shall exist a corresponding *YODA*-Project type. Each instantiation of a *YODA*-Project has to be of only one output type.
Priority | Risk	1 | XL

ID | Title	1.1.3.1 | **Supported Output Types, HTML**
Reference	-
Description	The set of given output types shall consist only of one entry, namely HTML documents.
Priority | Risk	1 | XL

ID | Title	1.1.3.2 | **Output extendibility, Markdown or XML**
Reference	R. 1.1.3.1
Description	The software architecture shall allow easy extensibility to support more *YODA*-Project output type in the future, namely XML or Markdown.
Priority | Risk	1 | XL

ID | Title	1.1.4.1 | **Multiple Project Instances, Project Types**
Reference	-
Description	The client shall be able to create an arbitrary number of *YODA*-Project instances, each of any wished supported *YODA*-Project type. All *YODA*-Project instances shall be completely independent from each other.
Priority | Risk	1 | L

ID | Title	1.1.5.1 | **Add *YODA*-Files to *YODA*-Projects**
Reference	R. 1.2.1.1, R. 1.1.1.1
Description	For a created *YODA*-File, the client shall have the ability to add it to an arbitrary number of *YODA*-Project instances, as long as their output type matches.
Priority | Risk	1 | XL

ID | Title	1.1.5.2 | **Same *YODA*-File, same *YODA*-Project**
Reference	R. 1.2.1.1, R. 1.1.1.1
Description	The client shall have the freedom to add a *YODA*-File to an arbitrary number of *YODA*-Project instances, as long as their output type match.

Priority | Risk	1 | XL

ID | Title	1.1.5.3 | **Order of *YODA*-Files**
Reference	R. 1.1.5.1
Description	The order of the *YODA*-Files in the final Output-Document shall be the same as the order in which they were added to the *YODA*-Project in the program code.
Priority | Risk	1 | S

ID | Title	1.1.5.4 | **Show *YODA*-Files in *YODA*-Project**
Reference	-
Description	For each *YODA*-Project, the client shall be able to print out all names of the *YODA*-Files contained in the *YODA*-Project to the console.
Priority | Risk	2 | S

ID | Title	1.1.6.1 | **Template, Conventions**
Reference	-
Description	For each *YODA*-Project, a template file shall be chosen. The Template file shall match the *YODA*-Project Type and consists of any arbitrary input, combined with predefined placeholder-tags that mark where *YODA* will insert the generated Content.
Priority | Risk	2 | M

ID | Title	1.1.6.2 | **Template, adding and changing**
Reference	R. 1.1.6.1
Description	Templates shall be either self-made with sticking to the predefined placeholder-tag convention, or chosen between a finite set of pre-created templates that come with downloading *YODA*-Library.
Priority | Risk	3 | S

ID | Title	1.1.6.3 | **Template, user created Templates**
Reference	R. 1.1.6.1
Description	The set of pre-created templates shall have the freedom to grow over time with clients submitting their Templates to the public GitHub repository of *YODA*.
Priority | Risk	3 | S

ID | Title	1.1.7.1 | **Render, *YODA*-Project, generate output**
Reference	-
Description	The client shall have the possibility to render a *YODA*-Project, meaning for every *YODA*-File and every *YODA*-Element, output that fitts the output-document type shall be produced and written into the template. If the output types requests, all files shall be correctly linked together.
Priority | Risk	3 | XL

ID | Title	1.1.7.2 | **Render *YODA*-Project Output Folder**
Reference	-
Description	All necessary output data shall be written into a folder of the client's choice, ready to get published.

Priority | Risk	3 | XL

ID | Title	1.1.7.3 | **Render *YODA*-Project as Preview**
Reference	-
Description	As a possible additional step before rendering the *YODA*-Project to the output folder, the client shall have the possibility to print the generated markup-content to the console in order to check and proof it before rendering the files to the disk.
Priority | Risk	3 | S

ID | Title	1.1.8.1 | **HTML Project, extends *YODA*-Project**
Reference	-
Description	*YODA*-Projects of Type HTML shall have additional, HTML Specific Attributes and Features, namely additional attributes for storing head-information and template-specific attributes like HTML-Header-Image or HTML-Footer-Content. All these additional values shall have the option to be set by the client.
Priority | Risk	1 | XL

3.1.2 File Related Requirements

ID | Title	1.2.1.1 | ***YODA*-File, Container of *YODA*-Elements**
Reference	-
Description	The client shall be able to create new *YODA*-Files, which serve as a container of *YODA*-Elements, Attributes and Features. Each *YODA*-File shall have a client chosen name for identification purposes as an attribute.
Priority | Risk	2 | L

ID | Title	1.2.2.1 | ***YODA*-File Types, File-Output Relation**
Reference	-
Description	For each supported output type, there shall exist a corresponding *YODA*-File type. Each instantiation of a *YODA*-File has to be of only one output type.
Priority | Risk	1 | XL

ID | Title	1.2.3.1 | **Add *YODA*-Elements to *YODA*-File**
Reference	-
Description	The client shall have the freedom to add *YODA*-Elements to an arbitrary number of *YODA*-File instances, as long as their output type match.
Priority | Risk	1 | XL

ID | Title	1.2.3.2 | **Order of *YODA*-Elements**
Reference	R. 1.2.3.1
Description	The order of the *YODA*-Elements in the final Output-Document shall be the same as the order in which they were added to the *YODA*-File in the program code.
Priority | Risk	1 | S

ID | Title	1.2.3.3 | **Allowed *YODA*-Elements in *YODA*-Files**
Reference	R. 1.2.3.1
Description	An *YODA*-Element can be added to a *YODA*-File an arbitrary number of times, at arbitrary

	places.
Priority | Risk	1 | L

ID | Title	1.2.4.1 | **Show *YODA*-Elements in *YODA*-File**
Reference	R. 1.2.3.1
Description	For each *YODA*-File, the client shall be able to print out all names of the *YODA*-Elements contained in the *YODA*-File to the console.
Priority | Risk	2 | S

ID | Title	1.2.5.1 | **HTML File, extends *YODA*-File**
Reference	-
Description	*YODA*-Files of Type HTML shall have additional, HTML specific attributes and features, namely a list of other *YODA*-Files which serve as its subpages in the menu. All these additional values shall have the option to be set by the client.
Priority | Risk	1 | XL

ID | Title	1.2.5.2 | **Subpages, layers**
Reference	-
Description	Every *YODA*-File of type HTML can serve as a subpage of another *YODA*-File, if and only if the *YODA*-File has no subpages itself. This implies that subpages only reach one single level deep.
Priority | Risk	3 | S

ID | Title	1.2.6.1 | **Rendering *YODA*-Files**
Reference	-
Description	Every *YODA*-File shall offer the functionality to render itself, meaning to render all its *YODA*-elements into the client-chosen template at the positions of the placeholder-tag and Output the proper formatted document to the output folder.
Priority | Risk	1 | XL

3.1.3 Element Related Requirements

ID | Title	1.3.1.1 | ***YODA*-Element Types, Element-Output Relation**
Reference	-
Description	For each supported output type, there shall exist a corresponding *YODA*-Element type. Each instantiation of an *YODA*-Element has to be of only one output type.
Priority | Risk	1 | XL

ID | Title	1.3.2.1 | ***YODA*-Element, represents Output-Snippet**
Reference	-
Description	Each *YODA*-Element shall be an abstraction of a supported feature of the Output-Document language, like Title, Table or Image.
Priority | Risk	1 | XL

ID | Title	1.3.2.2 | ***YODA*-Element, Types of Elements**
Reference	R. 1.3.2.1
Description	For the most important features of the output-document language, there shall exist corresponding *YODA*-Element-types representing that feature.

Priority | Risk	1 | XL

ID | Title	1.3.3.1 | ***YODA*-Element attributes**
Reference	-
Description	Each *YODA*-Element shall have the representation stored of how it formally looks in the output-document language, in order to later convert the abstraction of the *YODA*-Element to concrete output.
Priority | Risk	1 | XL

ID | Title	1.3.3.2 | ***YODA*-Element of type HTML special Features**
Reference	-
Description	In addition to normal *YODA*-Elements, the *YODA*-Elements of type HTML should have additional, HTML-Specific features like Name, Value pairs of Cascade Styling Commands to change the appearance of the element on the Output Page. Every *YODA*-Element of type HTML shall have a set of valid Names and offers the option to receive such Names from the client.
Priority | Risk	1 | XL

ID | Title	1.3.4.1 | **Render *YODA*-Elements**
Reference	-
Description	Each *YODA*-Element shall offer the functionality of rendering itself, meaning to convert itself into a proper text-based form to later fit the output-document.
Priority | Risk	1 | XL

ID | Title	1.3.4.2 | **Rendering, external resources**
Reference	-
Description	When *YODA*-Elements are rendered that rely on external resources like Images, the client shall need to store all these *YODA*-Files into a Folder called “Resources”, which lies in the previously chosen Output-Folder where the Documents get rendered to.
Priority | Risk	1 | XL

ID | Title	1.3.5.1 | **Different Types of *YODA*-Elements**
Reference	-
Description	There shall be two different types of Elements in YODA: The terminating *YODA*-Elements and the non-terminating ones. Each *YODA*-Element is either terminating or nonterminating, never both. Whether an *YODA*-Element is terminating or not depends on its accepted input values.
Priority | Risk	1 | XL

ID | Title	1.3.5.2 | **Terminating *YODA*-Elements**
Reference	R. 1.3.5.1
Description	A terminating *YODA*-Element shall be an *YODA*-Element that does not allow encapsulation in itself. A terminating *YODA*-Element can be encapsulated into a non-terminating *YODA*-Element, but not vice versa. A typical terminating *YODA*-Element is a text or image, in which no further *YODA*-Elements can be encapsulated.
Priority | Risk	1 | XL

ID | Title	1.3.5.3 | **non-terminating *YODA*-Elements**
Reference	R. 1.3.5.2
Description	Non-terminating *YODA*-Elements shall allow encapsulation, meaning they can receive other non-terminating or terminating *YODA*-Elements as its content to create several layers of encapsulation. A typical non-terminating *YODA*-Element would be a table, which can receive other non-terminating *YODA*-Element as cell-entries such as Lists, but also terminating entries like text or image.
Priority | Risk	1 | XL

ID | Title	1.3.5.4 | **Encapsulation layers**
Reference	R. 1.3.5.3
Description	Non-terminating *YODA*-Elements should offer encapsulation to an arbitrary number of levels. At the end of the encapsulation-chain, there always needs to be a terminating *YODA*-Element to end the encapsulation.
Priority | Risk	1 | XL

ID | Title	1.3.6.1 | **Wrapper-Functions**
Reference	-
Description	The client should not have to deal with manually creating instances of *YODA*-Element, like he has to when creating *YODA*-Files and *YODA*-Projects. Instead, the client should have the ability to use wrapper-functions that create the *YODA*-Element and return its instance.
Priority | Risk	2 | M

ID | Title	1.3.6.2 | **Concatenating Wrapper-Functions**
Reference	R. 1.3.6.1
Description	For every *YODA*-Element of any type and purpose, there should also exist one or more corresponding wrapper-function that creates and returns the *YODA*-Element. The arguments that the wrapper-functions should take directly correspond to the arguments defined in the *YODA*-Element it creates. The wrapper-functions should be able to directly be concatenated into each other in order to create different concatenation levels.
Priority | Risk	2 | M

ID | Title	1.3.7.1 | **Empty element - terminating**
Reference	-
Description	To force an ending of non-terminating elements, there shall be a special *YODA*-Element, called the empty element, which contains no content and serves just to terminate encapsulation.
Priority | Risk	2 | S

ID | Title	1.3.8.1 | **Text - terminating**
Reference	-
Description	The client should have the ability to create and add text-elements, which are just plain text encapsulated into an object to allow being encapsulated.
Priority | Risk	1 | L

ID | Title	1.3.8.2 | **HTML-Text, special attributes**

Reference	R. 1.3.8.1
Description	Text as a *YODA*-Element of Type HTML should additionally allow simple styling like making part of the text bold, italic, underline and more. The client should have the ability to encapsulate formatting commands to apply several of them to the same passage.
Priority | Risk	2 | S

ID | Title	1.3.8.3 | **Text, Tags as input**
Reference	R. 1.3.8.1
Description	The client's text input shall not have an impact on the output document's look, so *YODA* shall modify input text and exclude all words that are part of the output-document-types language.
Priority | Risk	1 | XL

ID | Title	1.3.9.1 | **Image - terminating**
Reference	-
Description	The client shall be able to add an image to his document that is either stored locally or on the web using a static URL. The client shall be obligated to state whether the given path points to a local or online image.
Priority | Risk	1 | L

ID | Title	1.3.10.1 | **Code Snippet - terminating**
Reference	-
Description	The client shall have the ability to insert his own code into the document, he should therefore have the ability to choose a file that contains a well-formatted code-snippet, which content will then be inserted into the *YODA*-File.
Priority | Risk	1 | L

ID | Title	1.3.10.2 | **Code Snippet - Conventions**
Reference	R. 1.3.10.1
Description	The code-snippet shall be obligated to follow certain conventions to guarantee that the snippet won't break the output file. The conventions are named in the *YODA*-Documentation for each supported output type. The snippet shall not be parsed or processed to prevent errors.
Priority | Risk	1 | XL

ID | Title	1.3.11.1 | **Button - terminating**
Reference	-
Description	The client shall have the ability to add buttons to his document. Every button can link to an external URL in the world wide web. The text on the button as well as the linked URL should be freely choosable by the client itself.
Priority | Risk	3 | S

ID | Title	1.3.11.2 | **HTML-Button - terminating**
Reference	R. 1.3.11.1
Description	In *YODA*, HTML-Buttons shall have the HTML-Specific ability to link to internal resources, like other *YODA*-Files in the same *YODA*-Project. To link to an internal *YODA*-File, the client shall state to which *YODA*-File the button shall link, and the button should then respond to

	whatever the *YODA*-File's URL is set to.
Priority | Risk	3 | S

ID | Title	1.3.12.1 | **Link - non-terminating**
Reference	-
Description	The client shall have the ability to add links to his *YODA*-Files. Every link can link to an external URL in the world wide web. Link are non-terminating, so every *YODA*-Element inside the link will be clickable and lead to the stated URL.
Priority | Risk	2 | M

ID | Title	1.3.12.2 | **HTML-Link - non-terminating**
Reference	R. 1.3.12.1
Description	In *YODA*, HTML-Links shall have the HTML-Specific ability to link to internal resources, like other *YODA*-Files in the same *YODA*-Project. To link to an internal *YODA*-File, the client shall state to which *YODA*-File the link shall link, and the link should then respond to whatever is stored behind that *YODA*-File's URL.
Priority | Risk	2 | M

ID | Title	1.3.13.1 | **Table - non-terminating**
Reference	-
Description	The client shall be able to insert tables with freely choosable content into his *YODA*-Files. The client provides two-dimensional data containing *YODA*-Elements, which will then be displayed in the individual cells of the table. Tables are non-terminating, meaning tables can have any content in it, even further tables.
Priority | Risk	2 | L

ID | Title	1.3.14.1 | **Title - non-terminating**
Reference	-
Description	The client shall have the freedom to create titles with certain strengths. Every *YODA*-Element inside the title-element, like text or buttons, shall be formatted big and strong like a title.
Priority | Risk	2 | L

ID | Title	1.3.15.1 | **List - non-terminating**
Reference	-
Description	The client shall be able to add lists to his files. Lists are either ordered or unordered, but never both at the same time. Lists are non-terminating, so each list element is a *YODA*-Element, so they can have any content in it, even further lists.
Priority | Risk	2 | L

ID | Title	1.3.16.1 | **Container- non-terminating**
Reference	-
Description	There might be cases where it is necessary to add more than one *YODA*-Element into an existing one, like adding a bunch of content into a table cell. Therefore, whenever one *YODA*-Element is requested but several shall be inserted, the client shall pack them into a container and add this single *YODA*-Element instead. Containers shall allow every *YODA*-Element to have multiple *YODA*-Elements on the same sublayer.

Priority | Risk	2 | S

ID | Title	1.3.17.1 | **Divisor - non-terminating**
Reference	-
Description	The client might wish to display content splitted on a horizontal line instead of just adding all the *YODA*-Elements vertically. Divisors shall give him the functionality to add content to different rows which will then be displayed vertically split from each other.
Priority | Risk	3 | S

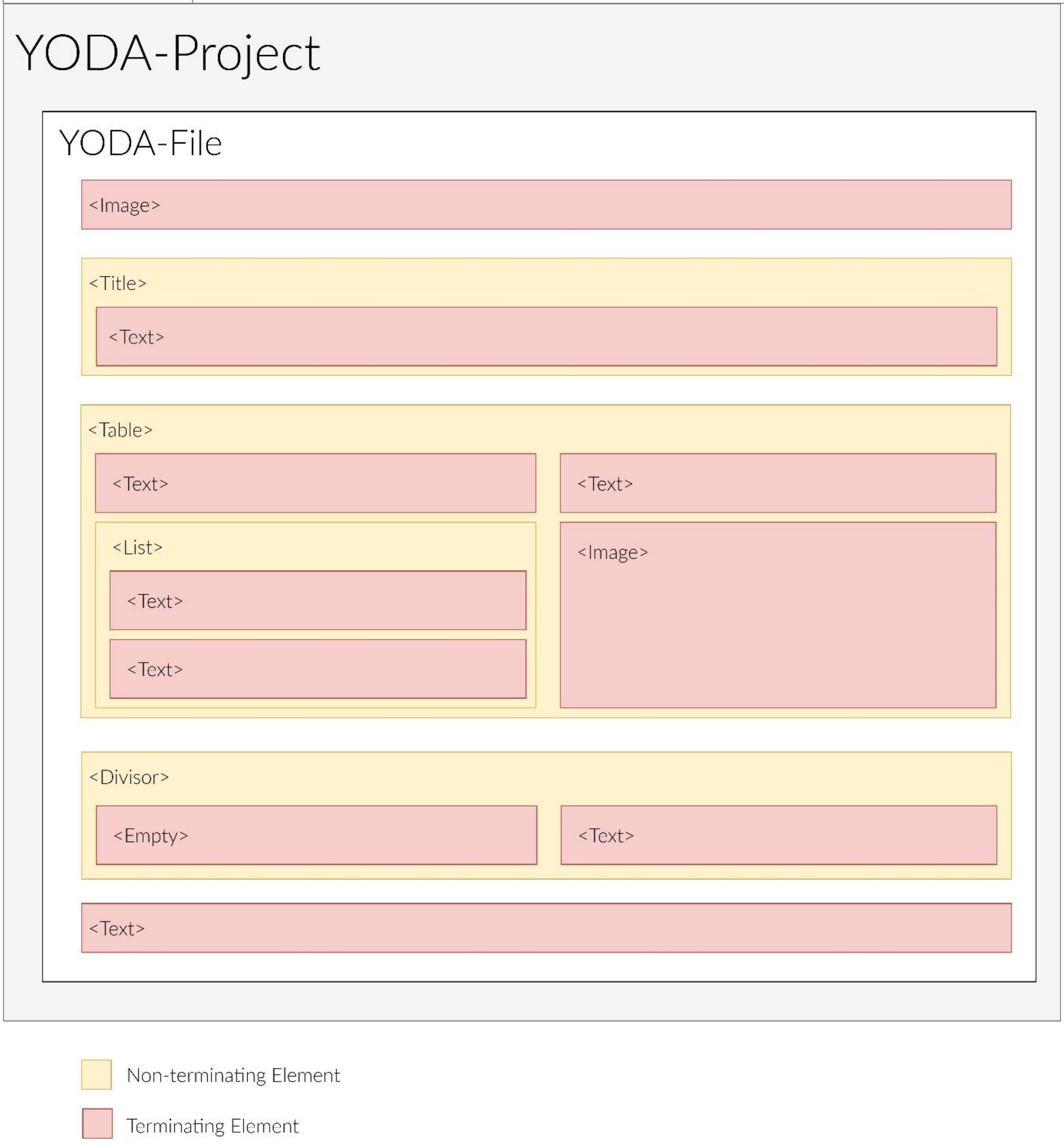


FIGURE 3: USAGE OF TERMINATING AND NON-TERMINATING ELEMENTS IN YODA

The diagram above shall illustrate how concatenation in YODA shall work with use of terminating and non-terminating elements.

3.2 Usability

This section includes requirements affecting usability like training time, task times or the language used.

3.2.1 Documentation

ID | Title	2.1.1.1 | **Documentation - learning**
Reference	-
Description	The *YODA*-Documentation should provide an easy start in creating convention-following output files without previous knowledge of the output-file structure. However, basic knowledge in Eiffel is implied. A client in the role academic with no background in the output files convention should be able to use all of *YODAs* functionality within 2 hours instructor-based training. Client in the role Administrator must be able to install the library and have an overview over in- and output data, which shall be provided after 1 hour of instructor-based training. Clients in the role designer should be able to modify existing output-type documents in order to automatically generate and place content in them within 0.5 hour of instructor-based training. A client in the role developer shall be able to extend *YODA* to other output-datas that are similar to XML in case of modularity of elements. Instructor-based training time for achieving this will take 8 hours.
Priority | Risk	1 | -

3.2.2 Training

ID | Title	2.2.1.1 | **Client Training**
Reference	-
Description	The following client-types must be able to use the system productively for their respective everyday work-life: ● Client, experience in Eiffel & HTML - after 0.5 days of training ● Client, experienced in Eiffel, no experience in HTML - after 1 day of training ● Client, no experience in Eiffel & HTML - after 3 days of training
Priority | Risk	1 | -

3.2.3 Task Times

ID | Title	2.3.1.1 | **Task Times**
Reference	-
Description	● For a trained client, creating a *YODA*-Project with one *YODA*-File containing no non-terminating *YODA*-Elements and at most 3 terminating *YODA*-Elements with no styling at all, shall take no longer than 30 minutes. ● For a trained client, creating a single *YODA*-Project with not more than 3 not interlinked *YODA*-Files each not containing more that 3 *YODA*-Elements with one nesting layer at most and no styling included, shall take no more than 2 hours. ● In general task times can differ in great extent depending on the complexity of the *YODA*-Project e.g. nesting layers, styling and interlinking of the *YODA*-Files as Subpages.
Priority | Risk	1 | -

3.2.4 Language

ID | Title	2.4.1.1 | **Language**

Reference	-
Description	The used language is English.
Priority | Risk	1 | -

3.3 Reliability

The Eiffel mechanisms such as static typing, assertions, automatic memory management and disciplined exception handling, enabling the *YODA-Programmers* to state correctness and robustness requirements, and enabling tools to detect inconsistencies before they lead to defects are the key factors that allow reliability.

3.3.1 Availability

ID | Title	3.1.1.1 | **Availability**
Reference	-
Description	*YODA* is used as a library without needing an internet connection in order to use it although it is recommended. The system shall be available for use at 24 hours a day, every day of the week. Since *YODA* shall be open source it shall be possible to add useful and correct functionality via GitHub. Clients can pull from this repository in order to add more functions.
Priority | Risk	1 | -

3.3.2 Error rate

ID | Title	3.2.1.1 | **Error rate**
Reference	-
Description	The quality of the input by the client is out of scope for this document. The first published version of the library shall have sufficient quality. This is defined by: 1. Not more than 2 errors per week 2. Not more than 2 patch severity errors per two weeks 3. Not more than 3 medium and low severity errors are per three weeks
Priority | Risk	2 | -

3.3.3 Error handling

ID | Title	3.3.1.1 | **Error handling**
Reference	3.2.1.1
Description	Error rate shall be as low as possible through using design by contract in Eiffel. In case of errors there shall be detailed error messages. Certain errors shall be handled automatically.
Priority | Risk	2 | -

3.3.4 Security

ID | Title	3.4.1.1 | **Security**
Reference	-
Description	*YODA* will not collect nor store any data of any clients.
Priority | Risk	1 | -

3.4 Performance Requirements

The golden mean between performance and efficiency like *Dr. Abstract* and *Mr. Microsecond* is hard to meet. But in order to be expandable the focus shall be on the side of architecture and abstraction to easily add other *features* and functionality.

3.4.1 Response Time

ID | Title	4.1.1.1 | **Response Time**
Reference	-
Description	Pure *YODA* code shall take an average of 10 seconds to compile and about 1 second to execute. The maximum shall lie between 20 to 30 seconds to compile and 5 seconds to execute.
Priority | Risk	1 | -

3.5 Maintainability

Maintainability is a design consideration concerning the ease with which *YODA* can be maintained once it is released and running. One aspect of Maintainability describes the facility of bug detection and bug fixing, in case of a malfunction or breakdown of *YODA*, referred to as corrective Maintenance. Another aspect is the capability of adapting *YODA* prompt and easily to changes in the environment such as a release of a new Eiffel version, known as adaptive maintenance. Moreover, maintainability involves the handling of perfective maintenance. The main focus lies on a highly extendable software architecture in order to respond to changed stakeholder requirements, both in terms of function and efficiency. A fourth aspect of maintainability deals with the preventative maintenance. This includes infrastructure to anticipate risks as well as the criterion of understandability achievable through consistent coding style and comprehensive documentation.

3.5.1 Corrective Maintenance

ID | Title	5.1.1.1 | **Bug classification**
Reference	-
Description	The *YODA*-Administrators classify bugs - that are local, without constraining the *YODA*-Main-Functionality into S (*Small Risk*) - that cause side-effects, without constraining the *YODA*-Main-Functionality into M (*Medium Risk*) - constraining the *YODA*-Main-Functionality partially without an imminent risk of a breakdown of YODA into L (*Large Risk*) - constraining the *YODA*-Main-Functionality profoundly with an imminent risk of a breakdown of YODA. into XL (*Extra Large Risk*)
Priority | Risk	1 | XL

ID | Title	5.1.1.2 | **Bug detection, Test coverage, Test disposability**
Reference	R. 5.4.1.1
Description	The *YODA*-Administrators should 1. check the *YODA*-Bug-Board on bugs reported by the *YODA*-Community once a week. 2. undertake the annual scheduled checks

	The *YODA*-Community should 1. write tests that covers 70% of the code. 2. should put the whole testing environment at the disposal of the community once the library is declared as open source.
Priority | Risk	2 | XL

ID | Title	5.1.1.3 | **Bug fixing**
Reference	R. 5.1.1.1
Description	The *YODA*-Administrators should fix bugs 1. of type XL within 1 2. of type L within 3 3. of type M within 7 4. of type S within 21 day(s) after reported on GitHub by the *YODA*-Community.
Priority | Risk	1 | XL

ID | Title	5.1.1.4 | **Bug communicating**
Reference	-
Description	The *YODA*-Administrators should set a *YODA*-Bug-Board up on their GitHub site. The *YODA*-Bug-Board consists of two lists, one contains supposed bugs reported by the *YODA*-Community, the other bugs proven by the *YODA*-Administrators. 1. The entries of the first list should consist of a bug index, a bug title, a short bug description referring to the observed limitations, the bug detection date. 2. The entries of the second list should consist of a bug index, a bug title, a short bug description referring to the caused limitations, the bug detection date, the bug classification and a bug fix duration prediction. After the fixing a bug fixing report should be added to the entry, which describes the conducted changes and the affected *YODA*-Methods.
Priority | Risk	1 | XL

3.5.2 Adaptive Maintenance

ID | Title	5.2.1.1 | **Compatibility with Eiffel**
Reference	-
Description	The *YODA*-Community should adapt the source code of the library, while keeping the same functionality, to new Eiffel versions that are not compatible with the current one within 1 month after the Eiffel versions release.
Priority | Risk	1 | XL

3.5.3 Perfective Maintenance

ID | Title	5.3.1.1 | **Commit incorporation**
Reference	-
Description	The *YODA*-Administrators should decide over the incorporation of committed code into the library within at most 21 days after commitment. An incorporation requires

	1. the new code being conform with the reviewed SRS 2. writing tests covering the new code 3. the new code passing all written tests An incorporation goes along with 1. incorporating the committed code into the source code 2. updating the *YODA*-Requirements-Checklist, the *YODA*-Documentation 3. making the correspondent tests accessible to the community by uploading them to GitHub.
Priority | Risk	3 | S

ID | Title	5.3.2.1 | **YODA-Requirements-Checklist**
Reference	-
Description	*YODA*-Administrators should write a *YODA*-Requirements-Checklist to facilitate keeping congruence between source code and requirements, and hold it up to date. One row for each requirement and one column for every update. The column header should consist of two lines, one for the update date and one for a brief summary of the changes done regarding the update. After every change in requirements the checklist must be updated by 1. recording the change in requirements a. add rows for new or changed requirements b. mark rows with out-of-date requirements 2. generating a new column with the date of the change 3. recording for each requirement if it is implemented in the current version After every change in code the checklist must be updated by 1. recording the change in code 2. generating a new column with the date of the change 3. recording for each requirement if it is implemented in the current version.
Priority | Risk	3 | S

ID | Title	5.3.2.2 | **YODA-Requirements-Checklist conformity**
Reference	R. 5.3.2.1
Description	The *YODA*-Administrators mustn't release code before it is at 100% conform to the *YODA*-Requirements-Checklist at state of release.
Priority | Risk	2 | L

ID | Title	5.3.3.1 | **Template collection**
Reference	-
Description	The *YODA*-Administrators should provide on GitHub a folder for output templates grouped into subfolders by the output file language. The administrators authorize the community to upload their templates.
Priority | Risk	3 | S

ID | Title	5.3.4.1 | **Software architecture, Continuity**
Reference	-

Description	The *YODA*-Programmer should apply extendibility to *YODA*s underlying architecture ensuring that small changes in the SRS induce only small changes in architecture. For this purpose, the Principle of Uniform Access should be taken into consideration.
Priority | Risk	2 | M

ID | Title	5.3.4.2 | **Software architecture, Single Choice Principle**
Reference	-
Description	The *YODA*-Programmer should apply the Single-Choice-Principle in respect of *YODA*-Projects, *YODA*-Files and *YODA*-Elements.
Priority | Risk	2 | M

ID | Title	5.3.4.3 | **Software architecture, Open-Closed Principle**
Reference	-
Description	The *YODA*-Programmers should structure the library in such a way that it is extensible for other Markup-language output formats, as well as additional functionality within an already existing Markup-language, by adding new classes, attributes and methods to the source code with heavy usage of already existing components, instead of changing or copy-pasting current code.
Priority | Risk	2 | M

ID | Title	5.3.4.4 | **Software architecture, Single Responsibility Principle**
Reference	-
Description	The *YODA*-Programmers should create every module in such a way, that it has responsibility over a single part of the functionality provided by the software, so that it has only one reason to change.
Priority | Risk	2 | M

ID | Title	5.3.4.5 | **Software architecture, Interface-Segregation Principle**
Reference	-
Description	The *YODA*-Programmers should split interfaces that are very large into smaller and more specific ones so that clients will only have to know about the methods that are of interest to them.
Priority | Risk	2 | S

3.5.4 Preventive Maintenance

ID | Title	5.4.1.1 | **Scheduled checks**
Reference	-
Description	The *YODA*-Administrators check *YODA* once a year by 1. running the entire test set over the source code in order to detect latent faults 2. studying the Class Diagram regarding to the architectural requirements defined in the SRS in order to detect out-of-date due to extensions and induce adaption 3. analysing the effects of the newest Eiffel version on *YODA*.
Priority | Risk	2 | S

ID | Title	5.4.2.1 | **Consistent Coding Style**
Reference	D3
Description	All *YODA*-Programmers in the team should apply the same coding style with respect to 1. Layout

	2. Names 3. Comments generating a consistent source code. The coding style should abide the Eiffel conventions described in "*Object-Oriented Software Construction*", see sections 4.2.1 and 4.2.2.
Priority | Risk	2 | S

ID | Title	5.4.2.2 | **Consistent Layout**
Reference	D3
Description	All *YODA*-Programmers should apply the same layout with respect to 1. Height and width 2. Indenting details 3. Spaces 4. Precedence and parentheses 5. Semicolons 6. Assertions generating a consistent source code.
Priority | Risk	1 | S

ID | Title	5.4.3.1 | **Support**
Reference	-
Description	The *YODA*-Administrators should provide 1. an open source code with comments conform to the SRS on GitHub 2. a *YODA*-Documentation on GitHub 3. answers within a month to unanswered questions on StackOverflow regarding the use of the library 4. an e-mail address for direct contact, with a response rate of at least 60% within 3 days for topics regarding bugs, extensions and unanswered problems that were posted on StackOverflow before.
Priority | Risk	2 | M

3.6 Design Constraints

Design constraints describe decisions made before building the system which have to be addressed during the building phase.

ID | Title	6.0.1.1 | **Non-interferring**
Reference	-
Description	*YODA* shall not interfere negatively with any other system- or operation.
Priority | Risk	1 | XL

3.6.1 Software language and Coding

ID | Title	6.1.1.1 | **Eiffel - Coding**
Reference	-
Description	All coding shall be done in Eiffel. At every point in time, all code shall be on the same

	version of Eiffel. The base Version used for this project is Eiffel 17.05, which is also the version the system shall be tested with.
Priority | Risk	1

ID | Title	6.1.1.2 | **External Libraries**
Reference	-
Description	*YODA* shall run on the client's system without installing or buying any additional libraries except the ones that standardly ship with Eiffel 17.05.
Priority | Risk	1 | -

ID | Title	6.1.1.3 | **Operating Systems used for Development**
Reference	-
Description	The development and all testing of *YODA* will happen on Windows and Mac OS.
Priority | Risk	1 | XL

3.6.2 Software process requirements

ID | Title	6.2.1.1 | **Introduction of new support output type**
Reference	R. 1.1.x.x, R. 1.2.x.x, R. 1.3.x.x
Description	If a new output type gets introduced, the following step-by-step instruction shall be applied: - add type specific *YODA*-Project class, regarding to the general *YODA*-Project class definition and the *YODA*-Project related requirements - add type specific *YODA*-File classes such that all *YODA*-File related requirements are met. - add type specific *YODA*-Element classes such that *YODA*-Element related requirements are met.
Priority | Risk	2 | S

ID | Title	6.2.1.2 | **General element-adding procedure**
Reference	-
Description	When adding a new *YODA*-Element class for a specific output type, a general class for that *YODA*-Element shall be added, if not existing. Only after the adding of the general *YODA*-Element-Class the output-specific element class shall be created.
Priority | Risk	2 | S

3.6.3 Development tools

ID | Title	6.3.1.1 | **EiffelStudio**
Reference	-
Description	For the development of *YODA,* the tool to be used shall be EiffelStudio 17.05.
Priority | Risk	1 | S

ID | Title	6.3.1.2 | **Version Control - GitHub**
Reference	-
Description	*YODA*'s versions shall be managed by one GitHub repository.
Priority | Risk	1 | S

3.6.4 Architectural and Design Constraints

ID | Title	6.4.1.1 | **Open Source**
Reference	-
Description	*YODA* shall be released under an open source free software licence GNU Library General Public License, version 2 (SPDX short identifier: LGPL-2.0)
Priority | Risk	2 | S

ID | Title	6.4.2.1 | **Template**
Reference	-
Description	At least one Template shall be available for *YODA* such that *YODA*-Elements can be added. All templates shall contain all placeholder-tags *YODA* uses for injection of the *YODA*-Elements into the output file.
Priority | Risk	1 | XL

ID | Title	6.4.3.1 | **Placeholder-Tags Convention**
Reference	-
Description	All Templates shall use the same placeholder-tags. The set of placeholder-tag shall be minimal but complete, which means it shall contain one and only one tag for every different Element supported per Markup-language. All placeholder-tags shall be listed in the *YODA*-Documentation
Priority | Risk	1 | XL

ID | Title	6.4.4.1 | **Library Constraints**
Reference	-
Description	*YODA* is a library, thus *YODA* shall NOT provide the functionality to edit and change *YODA*-Elements after instantiation since the *YODA*-Element do not exist until rendering. It's the client's task to instantiate the *YODA*-Element in the form in which it should finally look when rendering.
Priority | Risk	1 | XL

3.7 External Interfaces

EiffelStudio acts as the main interface to access all the functionalities of *YODA* such as creating new *YODA*-Project or adding content to them.

3.7.1 User Interfaces

The main client interface shall be EiffelStudio. All the functionalities of *YODA* can be accessed through EiffelStudio. Additional interfaces include templates on which the generated code can be displayed. Changes and additions to the displayed content shall be made in EiffelStudio.

3.7.2 Software Interfaces

Software interfaces include the templates provided by the *YODA*-Programmers but also the GitHub Repository of *YODA*. The purpose of the template is to match the *YODA*-Project Type and consist of any arbitrary input (R 1.1.6.1) so that the client can see and display the generated code but also create templates himself (R 1.1.6.2).

GitHub has the purpose of saving the pre-created templates and the client submitted templates so that it shall be possible to choose from different templates (R 1.1.6.3).

Another interface is the client provided folder named “resources” (R 1.3.4.2) through which *YODA* shall have access to the files used in the *YODA*-Project such as images or snippets.

3.7.3 Communications Interfaces

YODA does not contain any communication Interfaces. All communication from client to developer shall happen over GitHub and is not integrated into *YODA* itself. The client shall be provided with information about *YODA* via GitHub and the *YODA*-Documentation on there. This information exchange is also not integrated into *YODA* itself (R 3.6.1.1).

4 Supporting Information

4.1 Requirement index

4.2 Appendix

4.2.1 Class Diagram

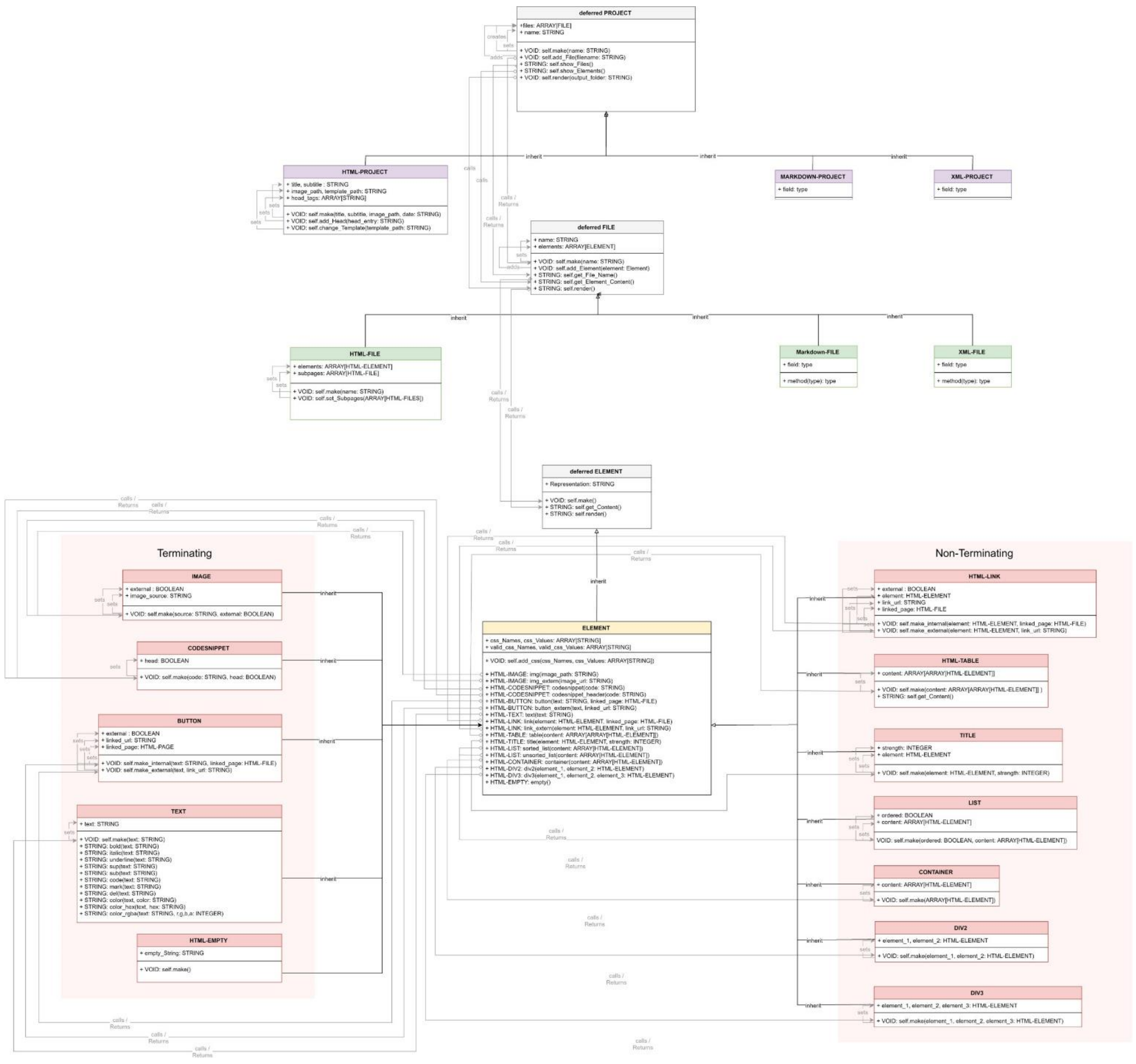

deferred PROJECT
+files: ARRAY[FILE]
+ name: STRING
+ VOID: self.make(name: STRING)
+ VOID: self.add_File(filename: STRING)
+ STRING: self.show_Files()
+ STRING: self.show_Elements()
+ VOID: self.render(output_folder: STRING)
HTML-PROJECT
+ title, subtitle : STRING
+ image_path, template_path: STRING
+ head_tags: ARRAY[STRING]
+ VOID: self.make(title, subtitle, image_path, date: STRING)
+ VOID: self.add_Head(head_entry: STRING)
+ VOID: self.change_Template(template_path: STRING)
MARKDOWN-PROJECT
XML-PROJECT
+ field: type
deferred FILE
+ name: STRING
+ elements: ARRAY[ELEMENT]
+ VOID: self.make(name: STRING)
+ VOID: self.add_Element(element: Element)
+ STRING: self.get_File_Name()
+ STRING: self.get_Element_Content()
+ STRING: self.render()
HTML-FILE
+ elements: ARRAY[HTML-ELEMENT]
+ subpages: ARRAY[HTML-FILE]
+ VOID: self.make(name: STRING)
+ VOID: self.set_Subpages(ARRAY[HTML-FILES])
Markdown-FILE
XML-FILE
+ method(type): type
deferred ELEMENT
+ Representation: STRING
+ VOID: self.make()
+ STRING: self.get_Content()
+ STRING: self.render()
ELEMENT
+ css_Names, css_Values: ARRAY[STRING]
+ valid_css_Names, valid_css_Values: ARRAY[STRING]
+ VOID: self.add_css(css_Names, css_Values: ARRAY[STRING])
Terminating
IMAGE
CODESNIPPET
BUTTON
TEXT
HTML-EMPTY
Non-Terminating
HTML-LINK
HTML-TABLE
TITLE
LIST
CONTAINER
DIV2
DIV3
inherit
calls / Returns
sets
adds
creates

FIGURE 4: FIRST PROTOTYPE OF A POSSIBLE YODA CLASSDIAGRAM

The Class Diagram above is a draft-version on how *YODA* might look like it is **not** the finished version.

4.2.2 Naming Convention

This naming convention summarizes the *Eiffel* naming convention described in “Object Oriented Software Construction”. It should serve the *YODA*-Programmers as an overview, but does not replace the mentioned reference book.

Letter case of Names	The programmers should write 1. *class* names and formal generic parameters in all uppercase characters 2. feature names, non-constant attributes, routines other than once functions, local entities and routine arguments in all lower-case characters 3. constant attributes and once functions with the first letter in uppercase and the rest in lowercase to make *class* texts consistent and readable.
Compound words	The programmers should write compound names by separating words by the underscore ("_") character to enhance readability.
Name	The programmers should choose names that are 1. meaningful - to enhance clarity by indicating the intent use of the bearer of the name 2. terse - to avoid exaggerated complexity by eliminating unneeded redundancies, including the applying of the composite feature name rule, 3. explicit - to enhance clarity by using full words, not abbreviations.
Grammatical categories	The programmers should 1. use nouns for *class* names, may use adjectives for *deferred classes* describing a structural property 2. apply the Command-Query separation principle for routine names 3. use verbs in the infinitive or imperative, possibly with complements for procedures 4. use nouns for non-boolean query names and adjectives (maybe in is_ form) for boolean queries, never use imperative or infinitive verbs for attributes and functions

4.2.3 Comment Convention

This comment convention summarizes the *Eiffel* comment convention described in “Object Oriented Software Construction”. It should serve the *YODA*-Programmers as an overview, but does not replace the mentioned reference book.

Header comments	The programmers should write in the source code for every routine a header comment with a one step further indentation than the start of the routine body. The header comment should be 1. informative by naming what a query returns, qualified noun for a non-boolean query, question form for a boolean query and imperatives or infinitives for a command 2. terse by saying what the routine does, not that it does it, applying the Command-Query Seperation principle, not paraphrasing type information or precondition's requirements
Feature clause header comments	The programmers should write in the source code for every feature a feature clause header comment on the same line as the keyword feature, characterizing the category the feature belongs to.
Indexing clauses	The programmers should write indexing clauses at the beginning of each *class*.
Non-header comments	The programmers should write non-header comments in the source code only if they provide additional information to the pre- and postconditions, the Invariants and the other forms of comments, needed to prevent confusion and errors. Non-header comments should be of a level of abstraction higher than the code it documents, summarizing its effect instead of paraphrase it.
Software entities	The programmers should write software entities like attributes or arguments in the source code between an opening and a closing quote.