DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Technical Drawing Annotation Tool

ANTARES, Desenvolvimento de Software LDA

2019 / 2020

1170551 Miguel Albergaria

isep Instituto Superior de Engenharia do Porto

Technical Drawing Annotation Tool

ANTARES, Desenvolvimento de Software LDA

2019 / 2020

1170551 Miguel Albergaria

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Informatics Engineering Degree

July 2020

ISEP Advisor: **Paulo Proença**

External Supervisor: **Jaime Rodrigues**

I am grateful to all those who contributed to my academic path and all those who have always believed and supported me in my decisions. To all my friends and family, thank you.

Miguel Albergaria

Acknowledgement

I would like to express my deep and sincere appreciation for Instituto Superior de Engenharia do Porto (ISEP), ANTARES Desenvolvimento de Software LDA and all those who have helped me through my academic path.

In particular, to Professor Paulo Proença and, both, Jaime Rodrigues and Hooshiar Zolfagharnasab for all the help and knowledge offered throughout the development of this project.

I would also like to give a special thanks to the Antares team for having me as their intern for the duration of this project and for providing me with all the tools, resources and feedback that I needed for a successful realization of this project.

Abstract

With markets becoming increasingly more competitive, organizations thrive to improve their efficiency by saving time and resources. The project detailed in this report, therefore, expects to introduce a new and innovative way organizations of the mechanical and metrology areas can keep their competitiveness levels high.

This project aims, through a desktop application, to facilitate the process that is the gathering of the PMI data of technical drawings, recurring, for that, to several deep learning algorithms. Throughout this report the process of development of this project is exposed, therefore, providing to the reader all the steps followed for the implementation of the solution.

Adding to this, this report also presents a guide of the project's final solution, allowing the reader to see its graphical interfaces, all its features and how these work through user interaction.

Finally, this document ends with a conclusion where the state of the final solution is presented, along with some of its limitations and a possible future development path. In addition, the author presents some insights about the development process, concluding the project as an overall success.

Keywords (Theme): Metrology, Mechanical Engineering, Technical Drawings, PMI Data

Keywords (Technologies): WPF, Python, gRPC

Index

1 Introduction 1

1.1 Context 1

1.2 Problem Description 1

1.2.1 Objectives 2

1.2.2 Approach 2

1.2.3 Contributions 3

1.2.4 Work planning 3

1.3 Report structure 5

2 State of art 7

2.1 Related works 7

2.2 Existing technologies 10

2.2.1 Organization Technological Context 10

2.2.2 Technologies for Desktop Application Development 10

3 Analysis and design of the solution 17

3.1 Analysis 17

3.1.1 Domain Model 17

3.1.2 Functional requirements 18

3.1.3 Non-functional requirements 21

3.2 Design 23

4 Solution implementation 29

4.1 Implementation Description 29

4.1.1 Code Implementation 29

4.1.2 Final Solution 37

4.2 Tests 41

4.2.1 Unit Testing 41

4.3 Solution evaluation 44

5 Conclusions 45

5.1 Objectives achieved 45

5.2 Limitations and future work 45

5.3 Final Assessment 46

References ***.......... 49***

Attachment A – UC 3 Sequence Diagram ***.......... 55***

List of Figures

Figure 1 - Scrum framework development process [3] 3

Figure 2 - Gantt diagram with project planning for February and March............ 4

Figure 3 - Gantt diagram with project planning for April............ 4

Figure 4 - Gantt diagram with project planning for May 4

Figure 5 - Gantt diagram with project planning for June 5

Figure 6 - HighQA Inspection Manager Interface [5] 7

Figure 7 - DISCUS Desktop Interface [7] 8

Figure 8 - InspectionXpert Interface [12]............ 9

Figure 9 - Solution Domain Model............ 17

Figure 10 - Use-case Model 18

Figure 11 - Containers Diagram - Logical View 24

Figure 12 - Technical Drawing Annotation Tool Component Diagram - Logical View 25

Figure 13 - The user drags a shape (Sequence Diagram) 28

Figure 14 – Application initial interface............ 39

Figure 15 – Importing a technical drawing............39

Figure 16 – Displaying a technical drawing............ 39

Figure 17 – Application interface after auto ballooning 40

Figure 18 – Shapes and annotations adorners............ 41

Figure 19 - The user starts the auto ballooning (Sequence Diagram)............ 55

List of Tables

Table 1 - Comparison of features and characteristics between Electron, Qt, JavaFX and WPF 14

Table 2 – Use Cases' Implementation State 37

List of Code Excerpts

Code Excerpt 1 - MVVM Setup (Bootstrapper) *30*

Code Excerpt 2 – Application Startup (App.xaml)........ *30*

Code Excerpt 3 - Dependency Injection Container (AppContainer) *31*

Code Excerpt 4 – View Model Typed Factory........ *32*

Code Excerpt 5 – Configuring application container (Bootstrapper)........ *32*

Code Excerpt 6 – IDTO interface........ *33*

Code Excerpt 7 – IDTOConvertable interface *33*

Code Excerpt 8 – ViewModelFactoryRepository........ *33*

Code Excerpt 9 – LayerRepository *34*

Code Excerpt 10 – Entity implementation *35*

Code Excerpt 11 – Publishing event using EventAggregator........ *35*

Code Excerpt 12 – Subscribing and publishing events using EventAggregator........ *36*

Code Excerpt 13 – View executing relay command........ *37*

Code Excerpt 14 – Relay command executing method *37*

Code Excerpt 15 – Unit test using xUnit and NSubstitute........ *42*

Code Excerpt 16 – Unit test using both helper classes *43*

Code Excerpt 17 – Unit testing using Xunit.StaFact *43*

Notation and Glossary

AAA	Arrange-Act-Assert
AI	Artificial Intelligence
API	Application Programming Interface
AS	Aerospace Standard
AWT	Abstract Window Toolkit
CAD	Computer-aided design
CMM	Coordinate Measuring Machine
CPU	Central Process Unit
CSS	Cascading Style Sheets
DRM	Digital Rights Management
DTO	Data Transfer Object
ERP	Enterprise Resource Planning
FAI	First Article Inspection
FURPS	Acronym representing a model for classifying software quality attributes
GD&T	Geometric dimensioning and tolerancing
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IoC	Inversion of Control
JDK	Java Development Kit
JVM	Java Virtual Machine
MTBF	Mean Time Between Failures
MVVM	Model-View-Viewmodel
OCR	Optical Character Recognition

OEM	Original Equipment Manufacturer
PDF	Portable Document Format
PMI	Product Manufacturing Information
PPAP	Production Part Approval Process
RIA	Rich Internet Application
SOLID	Mnemonic acronym for five design principles intended to make software designs more understandable, flexible and maintainable
SPC	Statistical Process Control
STA	Single-threaded apartment
SWT	Standard Widget Toolkit
TDP	Technical Data Package
TIFF	Tagged Image File Format
UC	Use Case
UI	User Interface
UWP	Universal Windows Platform
VMM	Virtual Machine Manager
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

1 Introduction

This document intends to detail the internship project “Technical Drawing Annotation Tool” carried out at ANTARES, Desenvolvimento de Software LDA.

On this chapter, the scope, objectives, approaches and contributions of the project are contextualized. The planning established for its realization and the structure adopted for this document are also described.

1.1 Context

This document was written for the Project/Internship (PESTI) class of the Informatics Engineering Degree (LEI) of Instituto Superior de Engenharia do Porto (ISEP). This curricular unit aims to apply the knowledge covered in the degree, as well as personal, interpersonal and social skills, for the design of software solutions, in a professional context.

The project proposed to the author was the development of an application in the metrology area, which facilitates the process of gathering data present in technical drawings of the mechanical engineering field.

Finding a viable solution for the market is a priority. However, there are inherent challenges to the realization of an elegant, robust and useful solution for the area of metrology and mechanics. Those challenges plus the degree of innovation of the idea behind the project, led the author to accept the challenge that is the realization of this internship project. Additionally, it is possible to explore and expand horizons in a wide range of new technologies, thus fostering the spirit of initiative and learning that the course encourages.

1.2 Problem Description

Currently, the process of retrieving and inserting the information present in a technical drawing, which is a 2D drawing that allows the visual communication of how something should be built [1], onto a computer is mostly manual.

For it, a person needs to understand the drawing to extract the information, both figures and GD&T text, and then manually insert the retrieved data into the computer, making it a time-consuming task, depending on the complexity of the object represented, that requires knowledge of the mechanical and metrology areas from whoever is doing the task.

1.2.1 Objectives

The proposed project aims to achieve the creation and development of a frontend application that will be capable of reducing the time required for the gathering and insertion of the data mentioned above, in the Problem Description section. It also must not imply any previous knowledge of the metrology and mechanical fields by its users.

The solution must be developed using good software engineering practices, like *SOLID Principles*, *MVVM architecture*, design patterns and testing, to improve maintainability and it is imperative to be of easy and intuitive use, as well as visually appealing.

1.2.2 Approach

For the realization of the project, considering the expected objectives, not only in the final solution but also in the development process, it was decided for the use of the Scrum *framework*.

Scrum is an agile *framework* for managing complex knowledge work [2], in other words, it is a flexible approach to the development process that replaces a programmed algorithmic approach with a heuristic one, with respect for people and self-organization to deal with unpredictability and solving complex problems [3]. The *framework* challenges assumptions of the traditional, sequential approach to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved [2].

A key principle of Scrum is the dual recognition that customers will change their minds about what they want or need and that there will be unpredictable challenges for which a predictive or planned approach is not suited, as such, it adopts the principle that the problem cannot be fully understood or defined upfront, and instead focuses on how to maximize the team's ability to deliver quickly, to respond to emerging requirements, and to adapt to evolving technologies and changes in market conditions [2].

By following this approach and its rules, throughout the development of the project, there were daily scrum meetings along with two-week sprints that ended in sprint reviews, where the work achieved in the past couple of weeks would be presented to all the teams as well as the product owner.

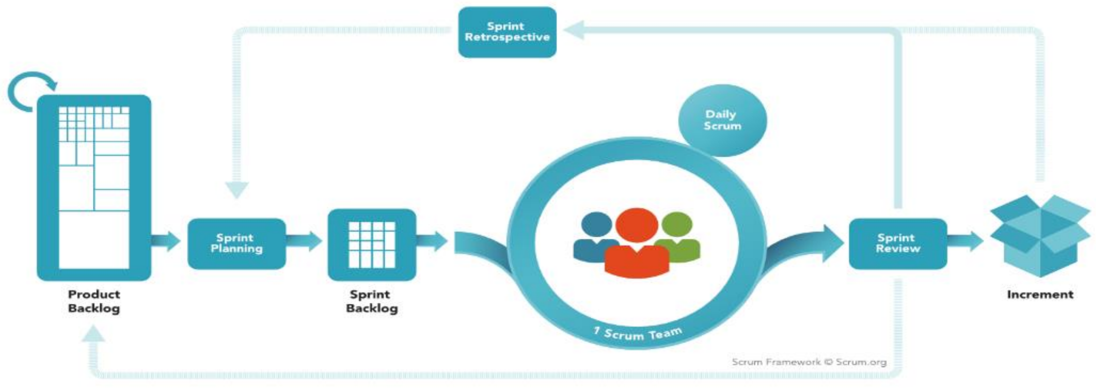


Figure 1 - Scrum framework development process [3]

1.2.3 Contributions

By achieving a working solution, this project will, as mentioned in the Objectives section, allow for a faster gathering of all the information present in a technical drawing without any knowledge needed from both the metrology and mechanical fields.

This will not only contribute to the organization and its clients, as well as to the mechanical and metrology areas as it will enable a new and innovative process of extracting the information of technical drawings, that relies on several deep learning algorithms as backend.

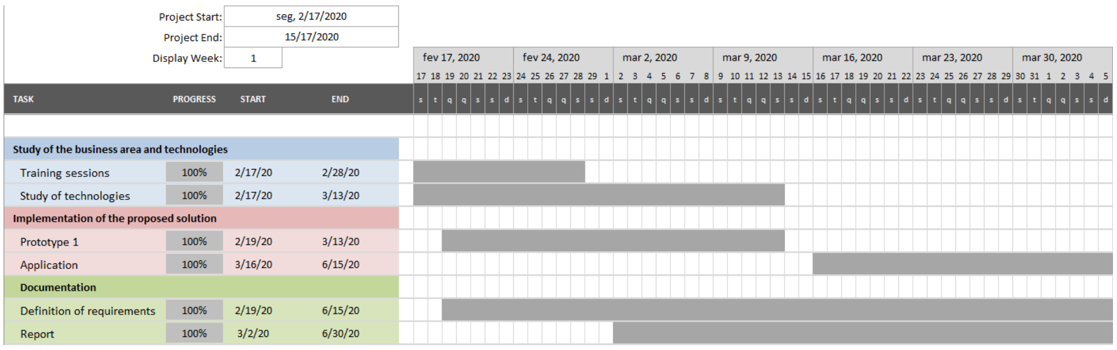
Using this solution organizations and individuals will be able to save time, which they can use for other tasks, thus increasing their efficiency. Along with giving a chance for less qualified workers to now handle this information extraction task, therefore allowing more qualified ones to focus on different more knowledge complex tasks.

1.2.4 Work planning

The development of this project was divided into two main iterations that overlapped. The study of the business area and technologies which lasted 26 days and the Implementation of the proposed solution that lasted for 118 days.

A Gantt diagram was prepared (Figures 2, 3, 4 and 5) containing the temporal sequence of all the tasks planned for the duration of the project. It should be noted that the weekly workload was of 25 hours.

February – March



Project Start:	seg, 2/17/2020
Project End:	15/17/2020
Display Week:	1

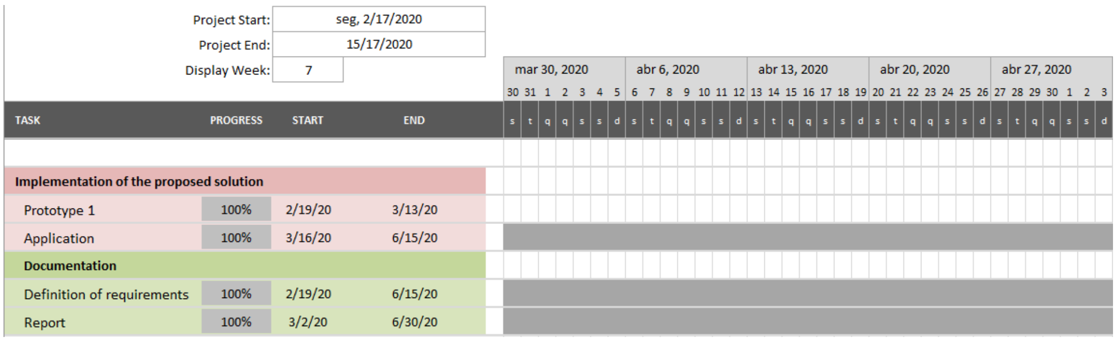
fev 17, 2020 · fev 24, 2020 · mar 2, 2020 · mar 9, 2020 · mar 16, 2020 · mar 23, 2020 · mar 30, 2020

17 18 19 20 21 22 23 24 25 26 27 28 29 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5

TASK	PROGRESS	START	END
Study of the business area and technologies			
Training sessions	100%	2/17/20	2/28/20
Study of technologies	100%	2/17/20	3/13/20
Implementation of the proposed solution			
Prototype 1	100%	2/19/20	3/13/20
Application	100%	3/16/20	6/15/20
Documentation			
Definition of requirements	100%	2/19/20	6/15/20
Report	100%	3/2/20	6/30/20

Figure 2 - Gantt diagram with project planning for February and March

April



Project Start:	seg, 2/17/2020
Project End:	15/17/2020
Display Week:	7

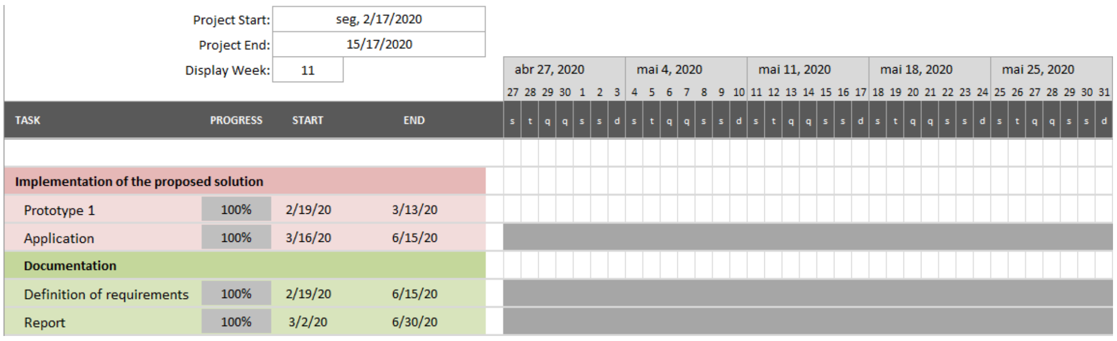
mar 30, 2020 · abr 6, 2020 · abr 13, 2020 · abr 20, 2020 · abr 27, 2020

30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3

TASK	PROGRESS	START	END
Implementation of the proposed solution			
Prototype 1	100%	2/19/20	3/13/20
Application	100%	3/16/20	6/15/20
Documentation			
Definition of requirements	100%	2/19/20	6/15/20
Report	100%	3/2/20	6/30/20

Figure 3 - Gantt diagram with project planning for April

May



Project Start:	seg, 2/17/2020
Project End:	15/17/2020
Display Week:	11

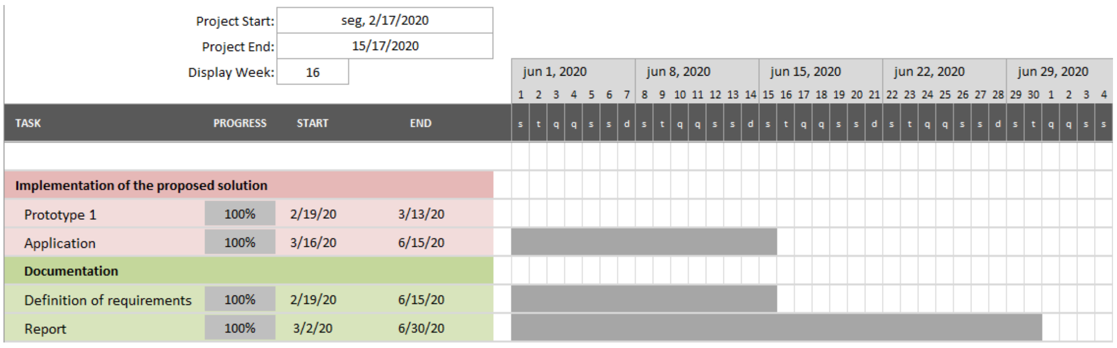
abr 27, 2020 · mai 4, 2020 · mai 11, 2020 · mai 18, 2020 · mai 25, 2020

27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

TASK	PROGRESS	START	END
Implementation of the proposed solution			
Prototype 1	100%	2/19/20	3/13/20
Application	100%	3/16/20	6/15/20
Documentation			
Definition of requirements	100%	2/19/20	6/15/20
Report	100%	3/2/20	6/30/20

Figure 4 - Gantt diagram with project planning for May

June



Project Start:	seg, 2/17/2020
Project End:	15/17/2020
Display Week:	16

TASK	PROGRESS	START	END
Implementation of the proposed solution			
Prototype 1	100%	2/19/20	3/13/20
Application	100%	3/16/20	6/15/20
Documentation			
Definition of requirements	100%	2/19/20	6/15/20
Report	100%	3/2/20	6/30/20

Figure 5 - Gantt diagram with project planning for June

1.3 Report structure

This last subchapter of the Introduction intends to briefly expose the structure and content of the document, which is divided into five main chapters: Introduction, State of art, Analysis and design of the solution, Solution implementation and Conclusions.

On the first chapter which is the Introduction, the scope, objectives, approaches and contributions of the project are contextualized, the planning established for its realization is presented and the structure adopted for this document is described.

Following the Introduction comes the State of art where related works are described along with their contributions to the current state of the field or fields in question. In this chapter it is also mentioned the existing technologies that could be used for the implementation of the solution, comparing their advantages and disadvantages, and justifying the chosen option.

On the third chapter, which can be divided into two main logical parts, the analysis and design of the solution are presented. On the analysis part, it is described the problem understanding process and its analysis along with a specification of the functional and non-functional requirements and the domain concepts. While on the design the global solution architecture is presented along with a justification of the choice in relation to the requirements. The use of software patterns and good practices are also covered throughout the chapter.

The fourth chapter is the Solution implementation where the details related to the implementation of the solutions recommended in the previous chapter are presented, along with the technologies and methodologies used. The implementation specificities are also described according to the development environment, platform and language chosen for development. At last, this chapter focus on the tests performed and the approach used for the solution assessment together with its results, which are compared to the planned ones.

On the final chapter, the conclusions reached through the development process are presented, demonstrating the comparison between the obtained results and the expected ones. It is also demonstrated the degree of achievement of the objectives described on the Introduction chapter, as well as an analysis of the work done, where the limitations of the work are shown along with possible future development paths. This chapter ends with an opinion from the author about the work developed.

2 State of art

On this chapter, it is described related works along with their contributions to the current state of the field or fields in question. It is also mentioned the existing technologies that could be used for the implementation of the solution, comparing their advantages and disadvantages, and justifying the chosen option.

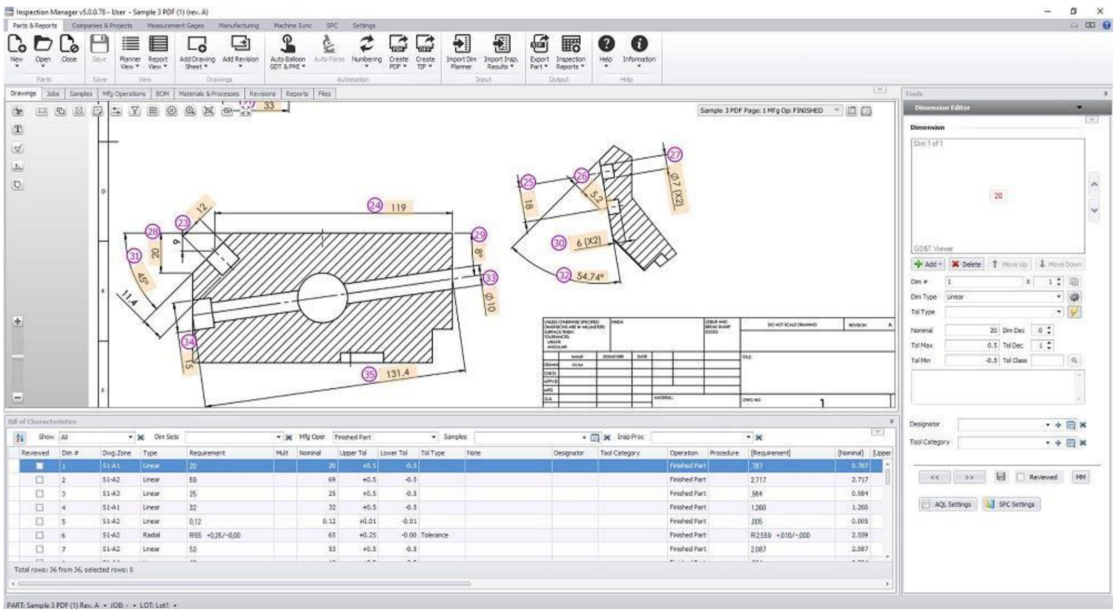
2.1 Related works

After an extensive market research, it was concluded that currently there are several other software solutions who offer tools, whose purpose is to help solve the identified problem and objectives. However, there are very few considerably identical applications.

On this subchapter the following software applications are presented: High QA Inspection Manager, DISCUS Desktop and InspectionXpert.

High QA Inspection Manager

High QA Inspection Manager is a software application developed by HighQA, whose purpose is to be an all-in-one manufacturing quality management system [4], thus encompassing several quality management modules.



Show: All | Dim Sets | Mfg Oper: Finished Part | Samples | Insp Proc

Reviewed	Dim #	Dwg.Zone	Type	Requirement	Mult	Nominal	Upper Tol	Lower Tol	Tol Type	Note	Designator	Tool Category	Operation	Procedure	[Requirement]	[Nominal]	[Upper
☐	1	S1-A1	Linear	20		20	+0.5	-0.5					Finished Part		.787	0.787	
☐	2	S1-A2	Linear	69		69	+0.5	-0.5					Finished Part		2.717	2.717	
☐	3	S1-A3	Linear	25		25	+0.5	-0.5					Finished Part		.984	0.984	
☐	4	S1-A1	Linear	32		32	+0.5	-0.5					Finished Part		1.260	1.260	
☐	5	S1-A2	Linear	0,12		0.12	+0.01	-0.01					Finished Part		.005	0.005	
☐	6	S1-A2	Radial	R65 +0,25/-0,00		65	+0.25	-0.00	Tolerance				Finished Part		R2.559 +.010/-.000	2.559	
☐	7	S1-A2	Linear	53		53	+0.5	-0.5					Finished Part		2.087	2.087	

Total rows: 36 from 36, selected rows: 0

PART: Sample 3 PDF (1) Rev. A • JOB: - • LOT: Lot1 •

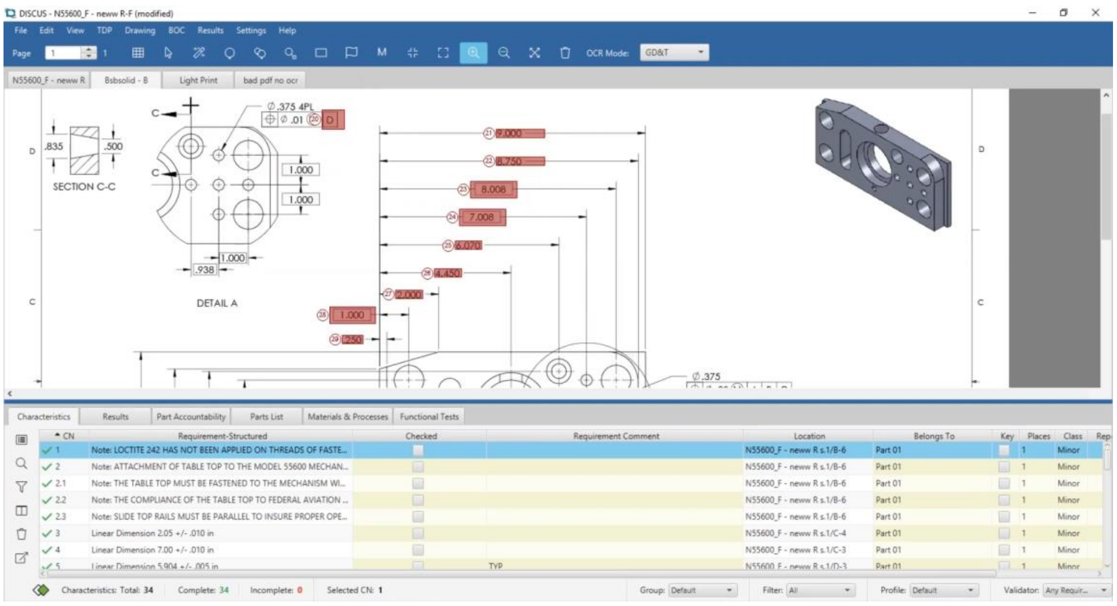
Figure 6 - HighQA Inspection Manager Interface [5]

This centralized *database-driven software* [4] allows for One-Click™ GD&T extraction and automatic ballooning using *AI*-driven *OCR*, inspection planning, data collection and organization, automated import of inspection results from *CMM*, *VMM* and other connected measurement equipment, shop floor data collection, creation of standard and custom forms and reports, integrated *SPC* tracking and reporting, user-access security management control and *ERP* integration [5].

Besides these mentioned features, Inspection Manager also provides an *API* for 3RD party integration, encrypted communications and flexible packages and licensing [4].

DISCUS Desktop

DISCUS Desktop is a *Windows* only *software* application developed by DISCUS Software Company [6], that enables its users to manage the *Technical Data Package* (*TDP*) and identify characteristics and requirements from 2D drawings and specification documents [6].



CN	Requirement-Structured	Checked	Requirement Comment	Location	Belongs To	Key	Places	Class
✓ 1	Note: LOCTITE 242 HAS NOT BEEN APPLIED ON THREADS OF FASTE...	☐		N55600_F - neww R s.1/B-6	Part 01	☐	1	Minor
✓ 2	Note: ATTACHMENT OF TABLE TOP TO THE MODEL 55600 MECHAN...	☐		N55600_F - neww R s.1/B-6	Part 01	☐	1	Minor
✓ 2.1	Note: THE TABLE TOP MUST BE FASTENED TO THE MECHANISM WI...	☐		N55600_F - neww R s.1/B-6	Part 01	☐	1	Minor
✓ 2.2	Note: THE COMPLIANCE OF THE TABLE TOP TO FEDERAL AVIATION ...	☐		N55600_F - neww R s.1/B-6	Part 01	☐	1	Minor
✓ 2.3	Note: SLIDE TOP RAILS MUST BE PARALLEL TO INSURE PROPER OPE...	☐		N55600_F - neww R s.1/B-6	Part 01	☐	1	Minor
✓ 3	Linear Dimension 2.05 +/- .010 in	☐		N55600_F - neww R s.1/C-4	Part 01	☐	1	Minor
✓ 4	Linear Dimension 7.00 +/- .010 in	☐		N55600_F - neww R s.1/C-3	Part 01	☐	1	Minor
✓ 5	Linear Dimension 5.904 +/- .005 in	☐	TYP	N55600_F - neww R s.1/D-3	Part 01	☐	1	Minor

Characteristics: Total: 34 Complete: 34 Incomplete: 0 Selected CN: 1 Group: Default Filter: All Profile: Default Validator: Any Requir...

Figure 7 - DISCUS Desktop Interface [7]

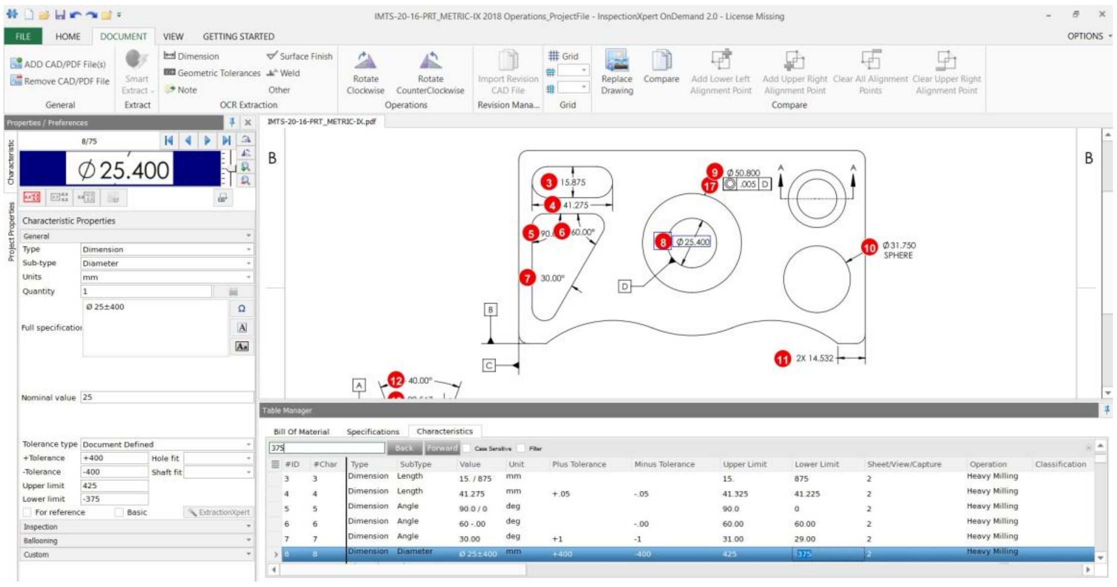
This *software* business model works by, besides the acquisition of the program, allowing the addition of features through several DISCUS add-ons [6], which are, at the time of the writing of this document, the following:

- **DISCUS OCR**: Enables text extraction from entire *PDF* and *TIFF* technical drawings, or just specific areas, using an *OCR* engine and DISCUS own RADAR™ technology [8].
- **DISCUS Results:** Enables the capturing and verification of inspection results against the drawing/model requirements [6].
- **DISCUS CMM:** Enables the importation of results from *CMM* Reports and the incorporation of the values into the ballooned drawing and inspection report [6].

- **DISCUS Planner:** Having a *TDP*, it enables the fast creation of a consolidated master list of requirements (i.e., the Bill of Characteristics) for the finished part [6] with support for *PDF* and Excel export [9].
- **DISCUS 3D:** Adds support for 3D *CAD* models, allowing for faster *CAD* models ballooning, using PMI data and algorithms that automatically identify characteristics [10]. Also allows for a faster creation of the Bill of Characteristics for these models [6].

InspectionXpert

InspectionXpert is a hybrid *online* inspection *software desktop* application developed by InspectionXpert, that offers productivity enhancement tools for manufacturing and quality inspections [11].



Characteristic Properties

General	
Type	Dimension
Sub-type	Diameter
Units	mm
Quantity	1
Full specification	Ø 25±400
Nominal value	25
Tolerance type	Document Defined
+Tolerance	+400
-Tolerance	-400
Upper limit	425
Lower limit	-375
Hole fit	
Shaft fit	
For reference	
Basic	
Inspection	
Ballooning	
Custom	

375

#ID	#Char	Type	SubType	Value	Unit	Plus Tolerance	Minus Tolerance	Upper Limit	Lower Limit	Sheet/View/Capture	Operation	Classification
3	3	Dimension	Length	15. / 875	mm			15.	875	2	Heavy Milling	
4	4	Dimension	Length	41.275	mm	+.05	-.05	41.325	41.225	2	Heavy Milling	
5	5	Dimension	Angle	90.0 / 0	deg			90.0	0	2	Heavy Milling	
6	6	Dimension	Angle	60 -.00	deg		-.00	60.00	60.00	2	Heavy Milling	
7	7	Dimension	Angle	30.00	deg	+1	-1	31.00	29.00	2	Heavy Milling	
8	8	Dimension	Diameter	Ø 25±400	mm	+400	-400	425	-375	2	Heavy Milling	

Figure 8 - InspectionXpert Interface [12]

This *software* focuses on improving the processes of ballooning drawings, collecting data and creating reports [13]. When it comes to the ballooning, InspectionXpert allows for automatic readings of GD&T callouts, with the use of *OCR*, creation of feature control frames, comparison and replacement of part revisions and adding, removing, editing and renumbering balloons [13]. For the collection part, this *software* allows for the importation of measurement results from any Excel or .txt file [13]. Lastly, when it comes to the report features, it supports the automated creation of AS9102, PPAP and FAI reports meeting the users OEM's requirements [11], as well as, importation and customization of Excel reports using a template editor and upload of inspection reports directly to Net-Inspect [11], which is an end-to-end supply chain and quality management *software* solution that can be securely accessed through any *web browser* [14].

Besides the mentioned features, InspectionXpert also includes *web* serviced flexibility, Adobe DRM, customizable *offline* access settings and instant *software* updates [11].

2.2 Existing technologies

Being the project a *desktop frontend* application, this chapter will focus on describing and comparing existing technologies that would allow the development of such applications. This subchapter also presents the organization technological context within the scope of the product where this project is inserted.

2.2.1 Organization Technological Context

Bearing in mind that the project in question is going to be part of the FARO CAM2 *software* it is important, before presenting the several development options, to contextualize it technologically.

FARO CAM2 Software is a 3D focused measurement platform, designed to enable users to efficiently fulfill their quality assurance and inspection tasks [15], having been developed in WPF (*Windows Presentation Foundation*).

Therefore, for an easier integration of the project with FARO’s current *software*, the application described throughout this document will be developed using the same technology, as per request of the organization. Nevertheless, in the following subchapter other possible technologies for the development of *desktop* applications are presented, along with a critical analysis and a comparison between them.

2.2.2 Technologies for Desktop Application Development

Electron

Electron (formerly known as Atom Shell) is an *open-source framework* [16] that allows the development of *cross-platform desktop* applications with *Web* technologies (JavaScript, HTML, and CSS), combining Chromium and Node.js at runtime to power up the applications, creating a *web* view in a *desktop* window [17].

Electron applications are composed of multiple processes. There is the "*browser*" process and several "*renderer*" processes. The *browser* process runs the application logic, and can then launch multiple *renderer* processes, rendering the windows that appear on a user's screen [16].

It was initially released July 2013, having been developed and maintained by GitHub, with its latest stable version, at the time of the writing of this document, released May 2020. Being the main *GUI framework* behind several notable *open-source* projects including Atom, GitHub Desktop, Light Table, Visual Studio Code and WordPress Desktop [16].

This *framework*, besides allowing the development of *desktop* applications using the mentioned *Web* technologies, also supports the use of the three main *frontend frameworks*: Angular, React and Vue.js. Making its entry level quite easy for people with experience using any of the mentioned technologies. When it comes to the *cross-platform* aspect, Electron supports macOS 10.10 (Yosemite) and above, Windows 7 and later and for Linux: Fedora 21, Debian 8 and Ubuntu 12.04 and newer versions [18].

But, despite the referenced advantages, Electron is known for its slow running capacity and high *resource* requirements [19] when compared to some other *desktop* application development technologies pushing away many developers from using it.

Qt

Qt is a *cross-platform* application development *framework* for *desktop*, *embedded* and mobile, written in C++ [20] and available under both *open-source* and commercial licenses [21]. Being currently used by some of the biggest companies of their areas, like Mercedes-Benz for its A-Class 2019 infotainment, LG for its LG's webOS [22], Autodesk for its desktop application Maya and Wolfram Mathematica.

Its initial release was in May 1995, having been, at first, developed by Troll Tech [23]. Throughout history Qt's parent company has changed several times being the current one The QT Company [24]. Also, its development, since 2011, follows an open governance model under The QT Project, allowing contributions from any member of the Qt ecosystem [25]. This has helped led it, through several updates, to what is considered by many the best designed C++ *GUI* application *framework*. Its last version, at the time of the writing of this document, was released May 2020 [26].

However, this status did not come by chance, but instead by its several well-structured features that allowed its growth in usability and, consequently, popularity. These are a complete and organized documentation, a fast performance, a vast and complete *API* and support of multiple languages, which are C++, Python and QML, as well as a support from the community of the languages: R and Go [27]. Currently, due to its growth, Qt has also developed another advantage, which is a big user base, that, in turn, facilitates the process of finding help and/or answers when in need. Relative to the *cross-platform* facet it supports

Windows 7 and above, macOS 10.13 and later, Linux and several of its distributions, like Ubuntu 18.04, Red Hat Enterprise Linux 7, and many more, Android 5.0 or higher, iOS 12 or above and tvOS 12 or above, watchOS 5 and later and UWP 10, which includes Windows 10, Windows 10 IoT devices, Xbox One and HoloLens [28].

But providing support for such a wide range of *platforms* also carries its disadvantage since each *platform* has its own characteristic visual stylings, designing a single *UI* for all platforms will, inherently, not look right when moved from machine to machine. Adding to this Qt uses a noticeable amount of memory and due to not being a C++ *library* and using a *metaobject compiler* has a more complex build process than some other *libraries* [29]. Because of this last point, *IDEs* and tools can flag Qt expressions as errors, slightly forcing the use of QtCreator as the ideal development tool [30].

JavaFX

JavaFX is a *software platform* made of and for Java, for creating and delivering *cross-platform desktop* applications, as well as rich *Internet* applications (RIAs) [31], having been created with the intention of replacing Swing, a *GUI widget toolkit* also for Java [32].

JavaFX was first released December 2008 by Sun Microsystems, which has since been acquired by Oracle [31], and since 2011 has been *open-source* as part of the OpenJDK, having been maintained and improved by the OpenFX community [33], that, at the time of the writing of this document, have made available its last version March 2020 [34].

When it comes to its advantages, this technology is known for its *cross-platform* aspect that supports Windows, Linux, Mac and mobile, both Android and iOS, as well as its ability to allow the development of the view side using FXML, an XML-based *user interface markup* language [35], CSS and HTML instead of Java. Besides these technologies, JavaFX also allows its applications development through a tool named Scene Builder, which enables the creation of the view side via a drag and drop functionality that generates the code automatically, therefore helping making the code process faster [36].

However, to allow the *cross-platform* between different computer *operating systems*, Java applications, and therefore JavaFX applications, run on an *instance* of a *software* called Java Virtual Machine (JVM), instead of directly on the computer [37], which, in turn, makes their *startup* slower because of the need for the device to start up the JVM before the application. In addition, JavaFX applications are known for their high memory consumption [38], as well as their slow performance when compared to some other similar technologies [39].

WPF (Windows Presentation Foundation)

WPF, which stands for Windows Presentation Foundation and previously known as Avalon, is a free and *open-source GUI framework* for developing Windows desktop applications [40].

This framework, originally developed by Microsoft, was released for the first time in November 2006 as part of the .NET Framework 3.0, but only became *open-source* December 2018 [40]. As of September 2019, WPF also got support by .Net Core 3.0 [41] and, at the time of the writing of this document, its last stable release was February 2020 [40].

Having been built from scratch and with no reliance on standard Windows controls, in most situations, [42] WPF has risen to be one, if not the most, used technology for Windows *desktop* applications development. This had to due to with all the features it provides when compared to some other similar same purpose technologies. These are its *vector graphics architecture*, which enabled infinite scaling with no *aliasing* [43], its powerful *data binding*, that coupled with an *MVVM* approach improved the applications testability, its support for both C# and Visual Basic .NET as code behind languages and XAML, which is a declarative *markup* language [44], as the view side language, its ability to make *user interfaces* for both Windows applications and *web* applications and its use of *hardware* acceleration for drawing the *GUI*, increasing performance [42]. Also due to its big user base and extensive documentation, the process of finding help online for any errors or doubts is relatively easy.

But, despite its vast number of advantages, WPF also suffers from some drawbacks that discourage its use in certain situations. These are its high memory usage, its steep learning curve, especially when using an *MVVM* approach, and the fact that it does not support the development of *cross-platform* applications.

Comparison

Due to the similarities between the presented technologies and for a better understanding of the characteristics and features of each, it was elaborated the Table 1.

The mentioned table exposes a comparison between each technology for several general characteristics, with each line having its own independent scaling. The aspects used for the comparison were: *open-source*, *cross-platform*, documentation quality, community support, technology maturity, memory usage, code reusability and application performance.

Table 1 - Comparison of features and characteristics between Electron, Qt, JavaFX and WPF

Characteristics/ Functionalities	**Electron**	**Qt**	**JavaFX**	**WPF**
Open source	Yes	Yes, and Commercial Licenses	Yes	Yes
Cross-platform	Yes	Yes	Yes	No
Documentation Quality	Decent	Very Good	Good	Very Good
Community Support	Low	High	Medium	Very High
Technology Maturity	Medium	Very High	High	High
Memory Usage	High	Low	High	Medium
Code Reusability	Medium	High	High	High
Application Performance	Low	High	Medium	High

Having concluded the comparison study between these desktop application development technologies, from the perspective of the author, WPF is the best suited technology for the development of the project.

This choice had to do with the generally higher scores of the chosen technology, the free aspect of it and with the fact that the project application will be integrated into an existing WPF application, therefore making the *cross-platform* irrelevant.

Other technologies

On this subchapter lie some other desktop application development technologies along with the reason why they were excluded from any chance of being the main technology for the proposed project.

- **Cocoa:** Is Apple's native object-oriented *application programming interface* (*API*) for its desktop operating system macOS [45]. It was excluded from the options for only supporting macOS applications development.

- **AWT (Abstract Window Toolkit):** Is Java's original *platform-dependent windowing*, *graphics*, and *user interface widget toolkit*, preceding Swing [46], thus it is outdated.
- **Swing:** Is a *GUI widget toolkit* for Java, developed to provide a more sophisticated set of *GUI components* than the earlier AWT [47]. Even developed as a replacement to AWT, Swing was also excluded for being outdated for the development of new desktop applications, due to the release of the newer technology JavaFX.
- **SWT (Standard Widget Toolkit):** Is a *graphical widget toolkit* for use with the Java *platform*, being an alternative to the AWT and Swing technologies [48]. As with Swing, SWT is also outdated when compared to JavaFX.
- **UWP (Universal Windows Platform):** Is a computing *platform* created by Microsoft and first introduced in Windows 10 [49]. It was excluded for only supporting Windows 10, when it comes to computers, and for only allowing *deployment* through Microsoft Store, which takes a share of the profit [17].
- **WinForms:** Is a free and *open source* [50] C# *GUI framework* for building Windows *desktop* applications. Being a .NET *wrapper* over Windows *user interface libraries*, such as User32 and GDI+ [51]. This technology was excluded for slowly being replaced by WPF, since both are very similar, with WPF being newer and more up to date with the current standards.
- **Win32:** Is the original *platform* for native C/C++ Windows applications that require direct access to Windows and *hardware*. This makes the Win32 API the *platform* of choice for applications that need the highest level of performance and direct access to *system hardware* [52]. This technology has the drawback of being of a lower level than the previous mentioned ones, therefore excluding it from an option, since it takes way longer to develop in it when compared to the others.
- **Xamarin.Forms:** Is an *open-source framework* for building iOS, Android, and desktop apps. Even though it supports the development of computer applications, this technology is focused on mobile development, therefore lagging in *desktop* development. Adding to this, for Windows, at the time of the writing of this document, it is only stable using UWP as the *backend platform*, which is one of the excluded technologies [53].

3 Analysis and design of the solution

On this chapter, which can be divided into two main logical parts, the analysis and design of the solution are presented. On the analysis part, it is described the problem understanding process and its analysis along with a specification of the functional and non-functional requirements and the domain concepts. While on the design the global solution architecture is presented along with a justification of the choice concerning the requirements. The use of software patterns and good practices are also covered throughout the chapter.

3.1 Analysis

On this section, the analysis of the solution is presented, consisting of the representation of the domain model and definition of the non-functional and functional requirements.

3.1.1 Domain Model

The system to be developed constitutes a desktop application for the gathering of the data present in technical drawings.

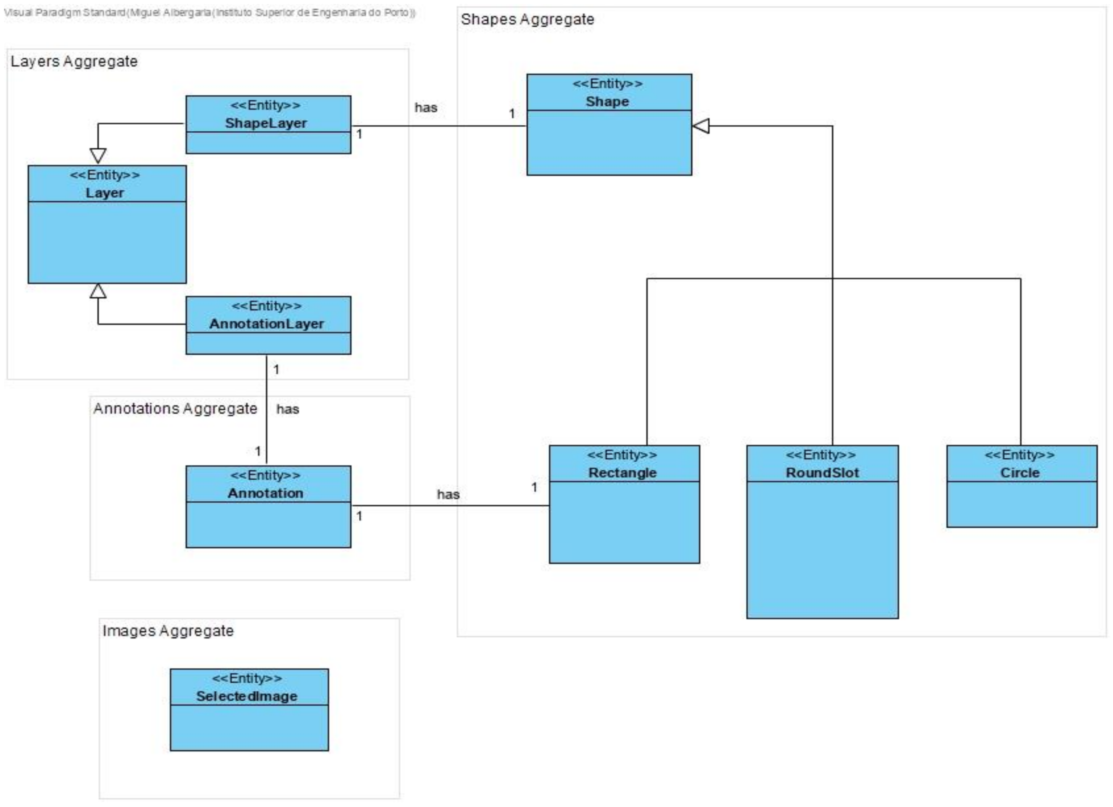


Figure 9 - Solution Domain Model

Therefore, for the extraction of the data, also known as *PMI*, users first need to import a technical drawing into the application, either as an ***image*** or a pdf, which will be converted to ***image***. The users then have the option to start the auto ballooning, which is the process of gathering the data present in the imported drawing. This will call the *backend* that, using deep learning algorithms, analyzes the ***image*** and returns the information it gathered, being each piece of it a ***layer*.** Each ***layer*** can either represent a ***shape*** or an ***annotation***, being, respectively, either a ***ShapeLayer*** or an ***AnnotationLayer***. Users can also add, remove or edit any ***layer***.

Annotations are text information, usually either GD&T callouts and/or numbers, and besides their content their also represented by a ***rectangle***, where the content is placed. ***Shapes*** can either be ***rectangles***, ***circles*** or ***round slots.***

3.1.2 Functional requirements

Bearing in mind the context of the intended solution, this subchapter involves defining the functional requirements of the project. These requirements are inherent to the features that the application is expected to offer and are represented in the use case format.

For the definition of the use cases, it was first identified the actors, which in UML specifies a role by a user or any other system that interacts with the subject [58], with the user being the only one, therefore having all use cases, which are exposed on Figure 10.

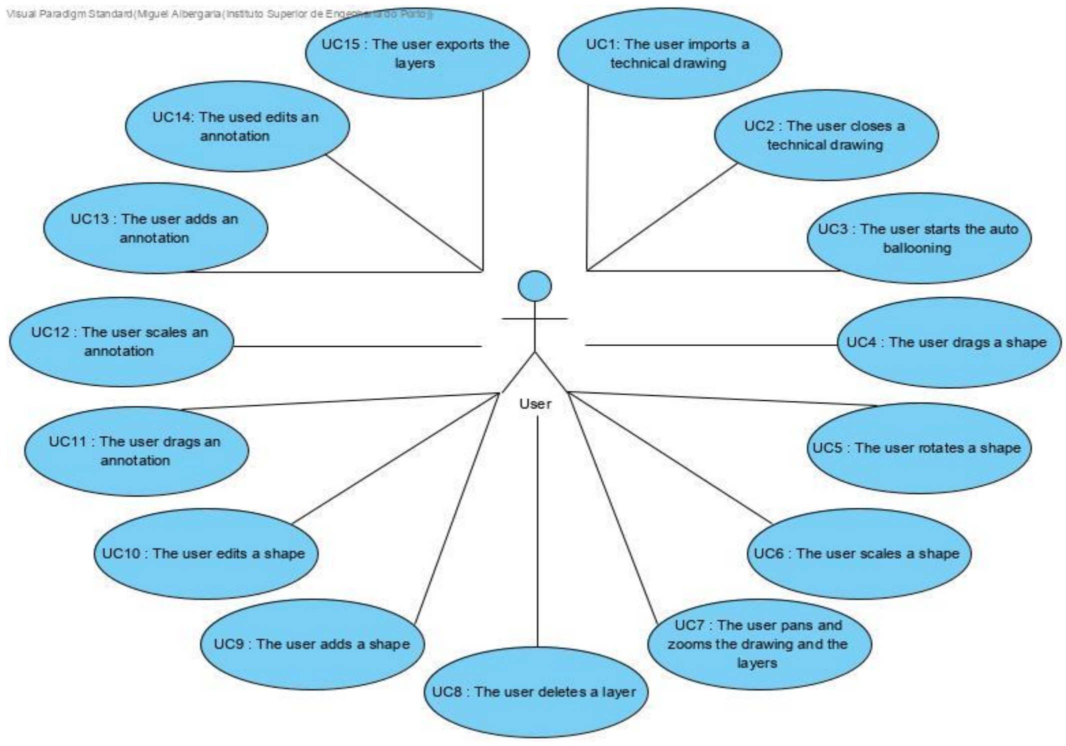

Visual Paradigm Standard(Miguel Albergaria(Instituto Superior de Engenharia do Porto))
UC15 : The user exports the layers
UC1: The user imports a technical drawing
UC14: The used edits an annotation
UC2 : The user closes a technical drawing
UC13 : The user adds an annotation
UC3 : The user starts the auto ballooning
UC12 : The user scales an annotation
UC4 : The user drags a shape
User
UC11 : The user drags an annotation
UC5 : The user rotates a shape
UC10 : The user edits a shape
UC6 : The user scales a shape
UC9 : The user adds a shape
UC7 : The user pans and zooms the drawing and the layers
UC8 : The user deletes a layer

Figure 10 - Use-case Model

- **Use Case 1 - The user imports a technical drawing**

  The user can import any technical drawing, either in image or pdf format, from his computer into the application. The imported drawing is then displayed and the auto ballooning option is unlocked and the export option locked if it was unlocked.

- **Use Case 2 – The user closes a technical drawing**

  The user can close a technical drawing. This removes both the drawing and the layers from being displayed. The auto ballooning option and export options are locked.

- **Use Case 3 – The user starts the auto ballooning**

  The user, after importing a technical drawing, selects the option to auto balloon it. The application sends the selected drawing to the *server* and it returns the shapes and annotations gathered from the drawing. Both shapes and annotations are then displayed on top of the drawing. The application unlocks the option to export the layers.

- **Use Case 4 – The user drags a shape**

  The user can drag shapes. The drag must be constrained to the image so that shapes cannot be dragged outside of it. The drag must be performed by interacting with the shape through mouse movements.

- **Use Case 5 – The user rotates a shape**

  The user can rotate some shapes, depending on the shape type. The rotation must be constrained to the image so that rotated shapes cannot be either fully or partially outside of the image. The rotation must be performed by interacting with the shape through mouse movements.

- **Use Case 6 – The user scales a shape**

  The user can scale shapes. Depending on the shape type the scaling can either be proportional or not. The scaling must be constrained to the image so that scaled shapes cannot be either fully or partially outside of the image. The scaling must be performed by interacting with the shape through mouse movements.

- **Use Case 7 – The user pans and zooms the technical drawing and layers**

  The user can zoom and pan the technical drawing image. And if the drawing has been ballooned then the layers that are displayed on top of it must also zoom and pan in synchrony with the drawing

- **Use Case 8 – The user deletes a layer**

  The user can delete any existing layer. This action should be able to be performed by either a press of a keyboard key or a button.

- **Use Case 9 – The user adds a shape**

  After auto ballooning the user can add any supported shape. For it, the user presses a button with the shape he wants to add and then clicks on the place, on top of the drawing, where he wants to add the shape. After adding, if the user continues pressing the mouse the shape should scale according to the mouse position, while always being constrained to the drawing.

- **Use Case 10 – The user edits a shape**

  The user through a menu can edit any shape color and opacity. He can also edit the visibility and name of the layer containing the shape.

- **Use Case 11 – The user drags an annotation**

  The user can drag annotations. The drag must be constrained to the image so that annotations cannot be dragged outside of it. The drag must be performed by interacting with the annotation through mouse movements.

- **Use Case 12 – The user scales an annotation**

  The user can scale annotations. The scaling must be constrained to the image so that scaled annotations cannot be either fully or partially outside of the image. The scaling must be performed by interacting with the annotation through mouse movements. When scaling the font size of the content should update according to the annotation size, so that it is always the biggest possible without cutting any text.

- **Use Case 13 – The user adds an annotation**

  After auto ballooning the user can add an annotation. For it, the user presses the create annotation button and then clicks on the place, on top of the drawing, where he wants to add it. After adding, if the user continues pressing the mouse the annotation should scale according to the mouse position, while always being constrained to the drawing.

- **Use Case 14 – The user edits an annotation**

  The user through a menu can edit any annotation content and opacity. He can also edit the visibility and name of the layer containing the annotation.

- **Use Case 15 – The user exports the layers**

  The user, after the image has been ballooned, can export the layers, saving all shapes positions into a *JSON* and all annotations positions and content into another *JSON*. This *JSONs* are then used for the retraining of the auto balloon deep learning algorithms.

3.1.3 Non-functional requirements

On this subchapter the analysis of the non-functional requirements of the project, which were raised by the Product Owner through several meetings, is presented, having been adopted the FURPS+ model for the classification of each requirement.

FURPS is an acronym representing a model for classifying *software* quality attributes [54], which embraces the non-functional categories of usability, reliability, performance and supportability [54]. This model subsequently evolved into FURPS+ having been added to the already existing requirements design, implementation, interface and physical constraints [55].

Usability

Usability is the attribute that assesses the *user interface* and aspects such as error prevention, design, help, documentation, consistency and standards [56].

Being this project a *desktop* application, aspects like *graphical interfaces* and fluidity are of great importance for viable user experience, with these characteristics being used with the intention of capturing new users, setting the application apart from other existing ones and facilitating its use.

The requirements defined for this attribute are:

- Simple and intuitive *graphical interfaces*, all based in a common theme between the application (i.e., colors, font sizes, fonts);
- Fluidity of the functionalities. There should be no visible lost in fluidity throughout the use of the application;
- All errors should be handled in a way that does not prevent the user from continuing using the application (i.e., avoid abrupt and unexpected closings).

Reliability

Reliability refers to the integrity, compliance and interoperability of the software, with the aspects considered being the frequency and severity of failures, possibility of recovery, forecasting possibility, accuracy and mean time between failures (*MTBF*) [56].

In the implemented solution, the following requirement is expected:

- In case of errors and/or failures, the application must remain operational.

Performance

Performance evaluates, as the name suggests, software performance requirements. Being able to use several aspects as a measure, among them: response time, memory consumption, *CPU* usage, load capacity and application availability [56].

With fluidity being a requirement and taking in consideration the stipulated objectives, performance assumes a high importance in the context of this project, since without it the applications usability and purpose would be defeated. Therefore, the requirement defined for this attribute is:

- Code should be optimized, and resources should be efficiently managed to allow for a smooth run of all visual functionalities.

Supportability

The supportability requirements group several characteristics, such as: testability, adaptability, maintainability, compatibility, configurability, installability, scalability, localization, among others [56].

And with the project being a desktop application so dependent on the *backend*, it is necessary for it to easily adapt to both *backend* changes as to users' feedback. Therefore, the requirements identified for this characteristic are:

- The system must be open to the addition of new functionalities;
- The code must respect defined standards, facilitating its maintenance (i.e., *SOLID principles*, dependency injection, *software* patterns);
- The application must be supported by Windows devices.

Design constraints

A design requirement, often called design constraint, specifies or restricts the design of a system [56]. This constraint defines the following requirement:

- The application must follow an *MVVM architecture*;

Implementation constraints

An implementation requirement specifies or restricts the code or construction of a system [56], therefore, defining the following requirements:

- The application must be developed using WPF;

- The application must use the *framework* Caliburn.Micro for the implementation of an *MVVM architecture*;
- The application must follow the *ViewModel* first approach;
- *Software* tests must be implemented.

Interface constraints

An interface constraint is a requirement to interact with an external item [55]. This constraint defines the following requirement:

- The project application must interact with the *backend server*, to allow for the auto ballooning of technical drawings and, subsequently, the retrieval of each the drawing data.

Physical constraints

Physical constraints specify physical limitations by the *hardware* used and may represent *hardware* requirements. This constraint defines the following requirements:

- Devices need access to the *internet* to be able to auto balloon the technical drawings;
- Devices must integrate the Windows *operating system* (version 7 or above [57]).

3.2 Design

This subchapter concerns the concepts adjacent to the business. In it the design of the final solution is presented, along with diagrams that were created following a junction of the C4 Model and the 4+1 architectural view model.

These diagrams are presented from a more general to more detailed order, with each having an explanation. It is also presented the design of some of the more complex use cases together with a detailed description of each.

Containers Diagram - Logical View

When it comes to the *containers* level, the final solution for the *desktop* application, as seen in the Figure 11, is made up of four *containers*.

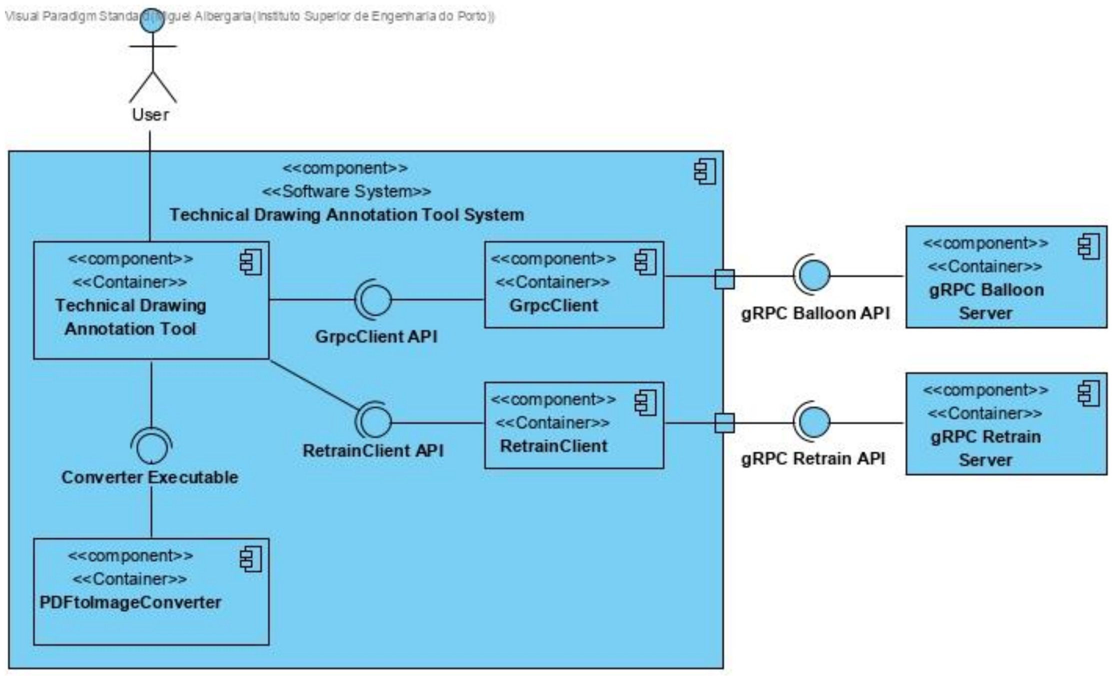


Figure 11 - Containers Diagram - Logical View

These are:

- **Technical Drawing Annotation Tool:** This is a .NET Core 3.1 WPF application, being the main *container* of the system. This is the *container* the user interacts with and, also, the *container* that interacts with the other three *containers*.
- **PDFtoImageConverter:** This is a python *executable* that, as the name suggests, allows for a conversion of PDFs to images. This *container* is necessary since the auto ballooning, which is the main feature of the application, only works with images. This *container* initially was not planned but had to be added because of the lack of free C# solutions that had a good quality conversion.
- **GrpcClient:** This is a .NET Core 3.1 application that acts as a *service* for the WPF application. This *container* is responsible for sending the technical drawing, in image format, to the **gRPC Balloon Server** and receiving the PMI of the drawing in JSON format. This *container* was created for a bigger level of separation between the WPF application and the gRPC server.
- **RetrainClient:** This is a .NET Core 3.1 application that acts as a *service* for the WPF application. This *container* is responsible for sending the PMI data of the technical drawing, after it has been edited through the WPF application, to the **gRPC Retrain Server**, so that the deep learning algorithms that perform the auto ballooning, through the **gRPC Balloon Server**, can be trained and, therefore, improved. This

container was created for a bigger level of separation between the WPF application and the gRPC server.

Of these, only the **Technical Drawing Annotation Tool** and the **PDFtoImageConverte**r are developed by the author.

Technical Drawing Annotation Tool Component Diagram - Logical View

With the Technical Drawing Annotation Tool *container* being the main *container* of the *system*, it is presented its component diagram, along with a detailed description of each *component*.

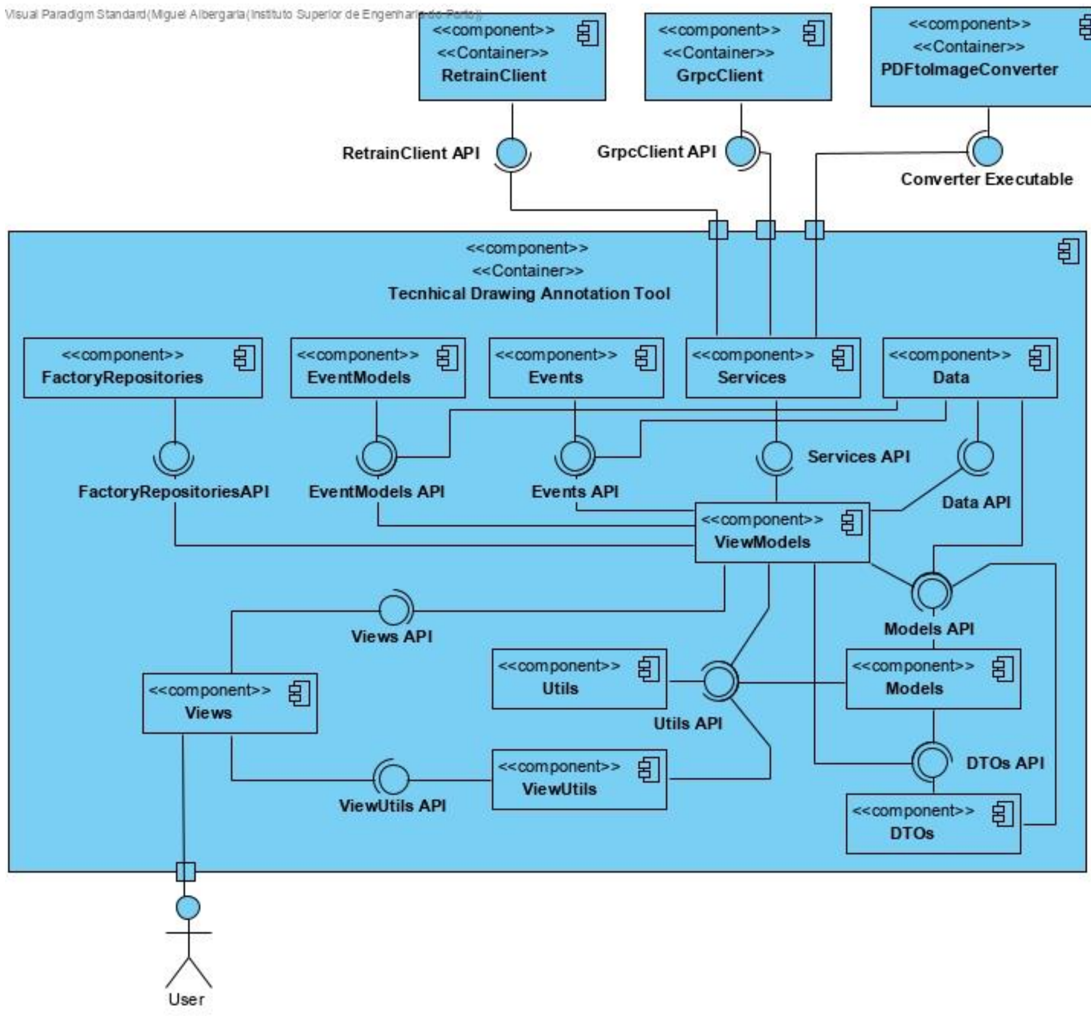


Figure 12 - Technical Drawing Annotation Tool Component Diagram - Logical View

- **Models:** This *component* represents the business models, which is the data being dealt with and that was presented in the Domain Model subchapter of the Analysis section. This *component* stands for the first M in ***M****VVM* (*Model-view-viewmodel*).

- **ViewModels:** This *component* is used for separating the **models** from the **views**, ensuring an independence of both, which facilitates the maintenance and testability. The **viewmodels** are responsible for manipulating the **models** and handling the logic behind the **views**, therefore existing one per **view**. Because of the **ViewModel** first approach, this *component* is also responsible for managing the life cycle of the **views**, which means that the **viewmodels** are, also, responsible for creating and deleting the **views**. This *component* stands for the VM in *MV**VM*** (*Model-view-viewmodel*).
- **Views:** This *component* is the one the user interacts with. It is the presentation of the data [59] and is active, meaning that it contains behaviors, events and data-bindings [59]. A window can be composed of many **views** or just one **view**. This *component* stands for the first V in *M**V**VM* (*Model-view-viewmodel*).
- **ViewUtils:** This *component* is responsible for all the utils used on the **views**, that is, all the reusable code that can be integrated into the **views**. It encompasses behaviors, adorners, validation rules, among others.
- **Utils:** This *component* is responsible for all the utility classes that can be reused throughout the application. Unlike the **ViewUtils**, this *component* is made for integration with C# code and not XAML.
- **Services:** This *component* is responsible for abstracting all the interactions of the Technical Drawing Annotation Tool container with other containers, therefore, improving maintainability and testability.
- **Data:** This *component* is responsible for storing all *instances* of data used in the application, with the use of the repository pattern. Besides that, this *component* is also responsible for publishing events anytime any **model** is added or deleted to a repository, so that the **viewmodels** can act accordingly and keep the **views** updated.
- **FactoryRepositories:** This *component* is a repository of **viewmodel** factories, using, as the name suggests, both the repository and factory patterns. This *component* allows for an easier creation of **viewmodels**, by taking advantage of the use of the mentioned patterns with the use of dependency injection.
- **Events:** This *component* represents **events** that can be published throughout the application whenever a certain action happens. This way, anytime an **event** is published all classes who subscribe to that event type will be notified, so that they can act accordingly.

- **EventModels:** This *component* works exactly like the **Events** one, with the difference that the **events** on this *component* have data in them, that can be accessed by the *subscribed classes*.
- **DTOs:** With the use of *JSONs* to communicate with the gRPC *servers*, this *component* facilitates the process of converting the *JSON* data into **models** and vice-versa, using, as the *component* name indicates, the *DTO* pattern.

Use Case 3 – The user starts the auto ballooning

The present use case concerns the main feature of the application, therefore, it is essential for it to be as efficient and organized as possible, both for the better user experience it accomplishes and the easier maintenance and testability. Figure 19, present on Attachment A, exposes the sequence diagram for the auto ballooning, which is only possible after importing a technical drawing.

This process is only triggered by user action, thus for it to start the **user** must press the auto ballooning button on the application menu. This will then publish an **AutoBalloonEvent** that, using an **EventAggregator**, notifies the **ShellViewModel**, which is the main *view model* of the application and the parent of the **MenuViewModel**. The use of events throughout the application is used to enable communications between *view models* in a low coupling manner, making it so that if a *view model* wants to notify or send data to another non child *view model* he doesn't have to follow the *view models* hierarchy, allowing for a change in the hierarchy without breaking any code.

After the **ShellViewModel** is notified it starts by getting the image source of the imported technical drawing and then, using the **ClientService** it communicates with the **GrpcClient** *container* to get the PMI information of the drawing. That information then comes in JSONs that are parsed and used to create *DTOs*, which, subsequently are converted into the model objects. In this case, the *DTOs* are used since the data that comes in the JSON is not enough for the creation of the models, therefore by moving the creation logic to the *DTO* object, it is maintained high cohesion on the parser, as well as, it is facilitated the process of the creation of the models, since it is now defined by the *DTO* instead of manually in the code.

Finally, the models, which are **Layer** instances, are added into the **LayerRepository**, which notifies, using events, the **LayersViewModel** and the **LayersMenuViewModel**. This *view models* then depending on the **Layer** type create their respective child *view models*, using *factories*, once again, promoting high cohesion and facilitating this creation process.

Use Case 4 – The user drags a shape

With the auto ballooning being done by deep learning algorithms, the occurrence of imprecisions or errors in the PMI data is notable. Thus, the application must allow its users to correct those mistakes. The dragging is, therefore, an essential feature for achieving the stipulated objectives. Figure 13 exposes the process for dragging a shape.

As with all use cases, this use case is only triggered by user action. It works by allowing direct manipulation [60] of the shapes through mouse movements, resorting to the use of the **DragElementBehavior**, that, as the name suggests, is a WPF *behavior* which encapsulates all the complex dragging logic into reusable code.

This way the **DragElementBehavior** can be integrated into the **ShapeView** allowing, both, the addition of logic to the *view* and the passing of data between the behavior and the view. The **ShapeView** is then able, through *data binding*, to pass that data to the **ShapeViewModel**, thus serving as an abstraction layer between the behavior and the *view model* and, therefore, allowing the dragging without any coupling to the *view model*.

By following this approach, the **DragElementBehavior** can be reused to also allow the drag of the annotations, which is another use case. The use cases for scaling and rotating also follow the same logic by using *behaviors*.

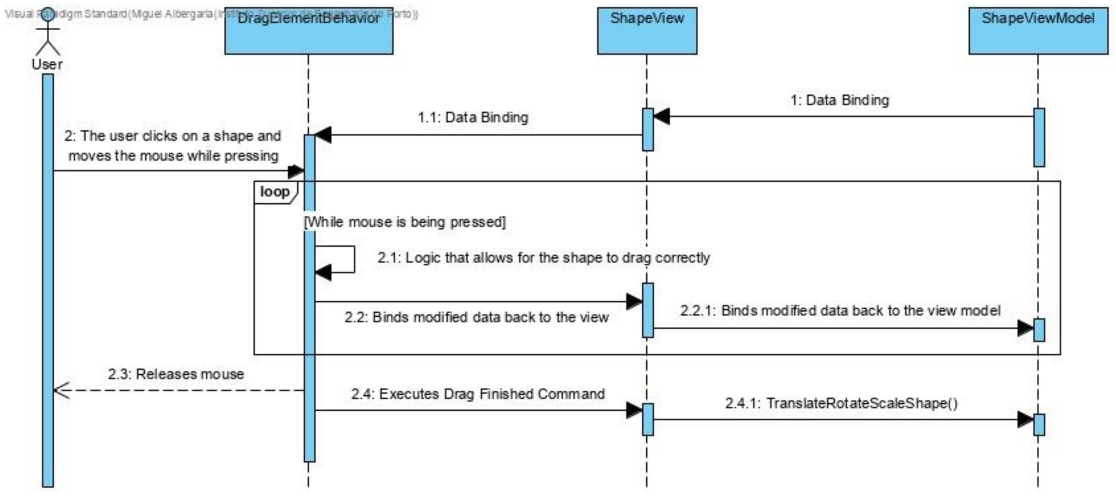

User
DragElementBehavior
ShapeView
ShapeViewModel
1: Data Binding
1.1: Data Binding
2: The user clicks on a shape and moves the mouse while pressing
loop
[While mouse is being pressed]
2.1: Logic that allows for the shape to drag correctly
2.2: Binds modified data back to the view
2.2.1: Binds modified data back to the view model
2.3: Releases mouse
2.4: Executes Drag Finished Command
2.4.1: TranslateRotateScaleShape()

Figure 13 - The user drags a shape (Sequence Diagram)

4 Solution implementation

On this chapter the details related to the implementation of the solutions recommended in the previous chapter are presented, along with the technologies and methodologies used. The implementation specificities are also described according to the development environment, platform and language chosen for development, and the implementation state of the use cases along with the graphical interfaces are also presented. At last, this chapter focus on the tests performed and the approach used for the solution assessment together with its results, which are compared to the planned ones.

4.1 Implementation Description

On this section, the implementation result of the final solution is exposed. The first subchapter of the section aims to explain important processes to the reader, along with code excerpts, while the second subchapter focuses on the graphical interface of the application and the state of the use cases.

4.1.1 Code Implementation

On this subchapter, several examples of the solution implementation are presented. It is explained the *MVVM* and *IoC container* setup, the use of the DTO pattern, entities implementation, *event aggregator*, use of *repositories* and the use of *relay commands* to improve testability.

This subchapter is tightly coupled with the Design section, since many of the practices exposed on this subchapter are mentioned and explained on the mentioned section.

MVVM Setup

Setting up the application architecture was a very crucial but simple process, mostly thanks to the used framework and its documentation.

For this process, it was used the *framework* Caliburn.Micro, as requested in the implementation constraints, which is a *framework* designed for building applications across all XAML *platforms* [61].

Using this *framework*, and following its documentation [62], it was just necessary to create a class, which is named **Bootstrapper**, to handle the application start (Code Excerpt 1) and then implement the **BootstrapperBase** class into the created class allowing for the setup of the *MVVM*. Using this base class it was, then, needed to override the **OnStartup** method (Code Excerpt 1, Line 10), setting up the ViewModel first approach and defining the **ShellViewModel** as the root *view model*. Caliburn.Micro follows a naming convention, so for the *MVVM* to work, *view model class* names need to end in ViewModel and *view class names* need to end in View. View model classes must also implement one of three base classes provided by Caliburn.Micro. These are:

- **Screen:** A view model should implement this class if it does not have any child view model.
- **Conductor<object>.Collection.OneActive:** A view model should implement this class if it can only have one child view model.
- **Conductor<object>.Collection.AllActive:** A view model should implement this class if it can have multiple child view models.

```
1  public class Bootstrapper : BootstrapperBase
2  {
3      (...)
4
5      public Bootstrapper()
6      {
7          Initialize();
8      }
9
10     protected override void OnStartup(object sender, StartupEventArgs e)
11     {
12         DisplayRootViewFor<ViewModels.ShellViewModel>();
13     }
14     (...)
15 }
```

Code Excerpt 1 - MVVM Setup (Bootstrapper)

With the *MVVM* setup, it was, then, just necessary to set the created *class* as the entry point of the application, which, again, was a very simple process that just required to add a few lines to the **App.xaml**, as seen in the Code Excerpt 2 from line 2 to 9.

```
1  <Application.Resources>
2    <ResourceDictionary>
3      <ResourceDictionary.MergedDictionaries>
4        <ResourceDictionary>
5          <local:Bootstrapper x:Key="Bootstrapper"/>
6        </ResourceDictionary>
7        (...)
8      </ResourceDictionary.MergedDictionaries>
9    </ResourceDictionary>
10 </Application.Resources>
```

Code Excerpt 2 – Application Startup (App.xaml)

IoC Container Setup

With the application able to run due to the *MVVM* setup, it was time to implement an *IoC* container for automatic *dependency injection*, using both Caliburn.Micro, to configure the application to use the *container*, and Castle Windsor [63], for the creation and setup of the container.

Starting with the creation of the container, it was created a custom container class, named **AppContainer** (Code Excerpt 3), that implements the Castle Windsor base container class, **WindsorContainer**. This custom class is responsible for registering all *services*, using *interfaces* for improved testability (Code Excerpt 3, Line 7), all view models and all view model factories, making them injectable. Both view models and view model factories are automatically registered through two custom methods (Code Excerpt 3, Line 12 and 19, respectively) that take advantage of naming conventions. For this, view models' names need to end with ViewModel, therefore, also following Caliburn.Micro convention, and view model factories' names need to end in ViewModelFactory.

```
1   public class AppContainer : WindsorContainer {
2     public AppContainer()
3     {
4       AddFacility<TypedFactoryFacility>();
5       Register(Component.For<ViewModelFactoryRepository>().LifeStyle.
Is(LifestyleType.Singleton),
6       (...)
7       Component.For<IPDFtoImageService>().ImplementedBy<PDFtoImageService>().
LifeStyle.Is(LifestyleType.Singleton));
8       RegisterViewModelFactories();
9       RegisterViewModels();
10    }
11
12    private void RegisterViewModelFactories()
13    {
14      List<Type> factories = new
List<Type>(Assembly.GetExecutingAssembly().GetTypes().Where(x => x.IsInterface &&
x.Name.EndsWith("ViewModelFactory")));
15      foreach (Type factory in factories)
16      { Register(Component.For(factory).AsFactory()); }
17    }
18
19    private void RegisterViewModels()
20    {
21      Register(Classes.FromAssembly(GetType().Assembly).Where(x =>
x.Name.EndsWith("ViewModel")).Configure(x => x.LifeStyle.Is(LifestyleType.Transient)));
22    }
```

Code Excerpt 3 - Dependency Injection Container (AppContainer)

But to register the factories, it was, first, necessary to implement them in a way they can support the creation of view models that have as parameters, both, registered dependencies and values, like strings, doubles, shape instances, among others. For this, it was used the Typed Factories feature of Castle Windsor (Code Excerpt 3, Line 4), which consists in creating an *interface*, for each view model, with a defined structure (Code Excerpt 4) that does not need to be implemented. This way to create a view model it is just necessary to call the factory interface create method with the values as parameters and Castle Windsor will automatically inject all the register dependencies the view model constructor has, along with the sent parameters.

```
public interface ILayersViewModelFactory
{
  LayersViewModel Create(Tuple<double, double> selectedImageDimensions, string
shapesAutoBallooningJSON, string textAutoBallooningJSON);

  void Release(LayersViewModel viewModel);
}
```

Code Excerpt 4 – View Model Typed Factory

Finally, with the *IoC* container setup it was necessary to add the container into the application, which was a simple process, thanks to the **Configure** and **BuildUp** methods (Code Excerpt 5, Line 7 and 14, respectively) that can be overridden in the **Bootstrapper** class.

```
1   public class Bootstrapper : BootstrapperBase
2   {
3       private AppContainer _container;
4
5       (…)
6
7      protected override void Configure()
8      {
9           _container = new AppContainer();
10     }
11
12     (...)
13
14     protected override void BuildUp(object instance)
15     {
16     instance.GetType().GetProperties()
        .Where(property => property.CanWrite && property.PropertyType.IsPublic)
        .Where(property => _container.Kernel.HasComponent(property.PropertyType))
        .ForEach(property => property.SetValue(instance,
_container.Resolve(property.PropertyType), null));
17     }
18  }
```

Code Excerpt 5 – Configuring application container (Bootstrapper)

DTO pattern

The DTO pattern, as mentioned in the Design section, was implemented, to improve the use of the *SOLID principles*, through the *single-responsibility principle*, as well as, to facilitate the creation of model objects and vice-versa.

This pattern was implemented using two *interfaces*, **IDTO** (Code Excerpt 6) and **IDTOConvertable** (Code Excerpt 7). The first *interface*, **IDTO** (Code Excerpt 6)**,** is responsible for allowing DTOs to convert into models.

```
public interface IDTO
{
  public IDTOConvertable ToModel();
}
```

Code Excerpt 6 – IDTO interface

While the second *interface*, **IDTOConvertable** (Code Excerpt 7), is responsible for allowing models to be converted into DTOs.

```
public interface IDTOConvertable
{
  public IDTO ToDTO();
}
```

Code Excerpt 7 – IDTOConvertable interface

Repositories

Throughout the application there are two types of repositories. The first, seen on Code Excerpt 8, acts as an index of *objects*, *classes* or *interfaces*, and never changes throughout the running of the application. Repositories of this type are done so that if any objects need multiple similar dependencies, they can just get the repository injected and access those dependencies through it. This way keeping the constructors clean and organized.

```
public class ViewModelFactoryRepository
{
 public IMenuViewModelFactory MenuViewModelFactory { get; set; }
 (...)
 public IAnnotationViewModelFactory AnnotationViewModelFactory { get; set; }
}
```

Code Excerpt 8 – ViewModelFactoryRepository

The second type, only used on Code Excerpt 8, acts as an *in-memory database*, allowing for the storing of data. This type of repository allows for adding, removing and getting methods and uses events to notify the application of certain actions.

```
public class LayerRepository
{
  private readonly IEventAggregator events;
  private readonly List<Layer> layers;

  public LayerRepository(IEventAggregator events)
  {
    this.events = events;
    layers = new List<Layer>();
  }

  public void AddLayer(Layer layer)
  {
    (...)
  }

  public void DeleteLayer(Layer layer)
  {
    (...)
  }

  public Layer GetLayerById(int id) {
    (...)
  }

  (...)

}
```

Code Excerpt 9 – LayerRepository

Entities

With all the *entities*, represented in the Domain Model, having an Id attribute and the need for some of the same methods, it was created an **Entity** class (Code Excerpt 10).

This class being *abstract*, is implemented into the models and allows for code reusability. This class also provides an implementation of the **INotifyPropertyChanged**, which is a WPF *interface*, allowing for any class that implements the **Entity** class to be able to, using the **NotifyPropertyChanged** (Code Excerpt 10, Line 12) method, notify of property changes.

View models then just need to subscribe to the models **PropertyChanged** property provided by the **Entity** class (Code Excerpt 10, Line 10) to receive notifications from the models.

```
public abstract class Entity : INotifyPropertyChanged
{
  public int Id { get; }

  public Entity(int id)
  {
    Id = id;
  }

  public event PropertyChangedEventHandler PropertyChanged;

  protected void NotifyPropertyChanged([CallerMemberName] string propertyName = "")
  {
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
  }

  (…)

}
```

Code Excerpt 10 – Entity implementation

EventAggregator

With Caliburn.Micro providing an event aggregator, the process of publishing and subscribing to events was very straightforward.

For it, an object just needs to receive the **EventAggregator** as a parameter, which is injected thanks to the use of the *IoC* container, and then using it subscribe to (Code Excerpt 11, Line 4) and/or publish events (Code Excerpt 12, Line 12).

```
1   private async void HandleSelectedItem()
2   {
3           (...)
4     await events.PublishOnUIThreadAsync(new OpenImageEvent());
5     (...)
6   }
```

Code Excerpt 11 – Publishing event using EventAggregator

When subscribing to an event it is also necessary for the object to implement an **IHandle<T>** *interface*, being **T** the event the object is subscribing to. This *interface* then forces the implementation of a **HandleAsync** method that has the **T** event has a parameter (Code Excerpt 12). This method is then executed every time an event of the **T** type is published. An object can subscribe to has many events as it needs by just implementing multiple **IHandle** *interfaces* and different object can subscribe to the same event.

```
public class MenuViewModel : Screen, IHandle<UnlockAutoBalloonEvent>,
IHandle<UnlockExportEvent>, IHandle<LockExportEvent> {

  public MenuViewModel(IEventAggregator events)
  {
    this.events = events;
    this.events.SubscribeOnPublishedThread(this);
    (...)
  }

  (…)

  // Handle unlocking Auto Ballooning
  public Task HandleAsync(UnlockAutoBalloonEvent message, CancellationToken
cancellationToken)
  {
    (...)
  }

  // Handle unlocking Export
  public Task HandleAsync(UnlockExportEvent message, CancellationToken
cancellationToken)
  {
    (...)
  }

  // Handle locking Export
  public Task HandleAsync(LockExportEvent message, CancellationToken cancellationToken)
  {
    (...)
  }
}
```

Code Excerpt 12 – Subscribing and publishing events using EventAggregator

RelayCommand

With testability being an important process of *software* development, it was implemented a custom command, using the WPF **ICommand** *interface*, named **RelayCommand**. This command allows the view to execute methods from its view model, in a way that allows those methods to be tested.

For this, view models simply need to have a public relay command that calls the wanted method and views just have to call that command. Relay commands, therefore, work as an abstraction between the views and the viewmodels.

An example of the **RelayCommand** use can be seen in the Code Excerpt 12 and 13, with the view calling the **LoadEditingFeaturesCommand** relay command(Code Excerpt 12, Line 3), and the command, subsequently, calling the **LoadEditingFeatures** method (Code Excerpt 13, Line 7).

```
1  <i:Interaction.Triggers>
2    <i:EventTrigger EventName="KeyUp">
3      <i:InvokeCommandAction Command="{Binding LoadEditingFeaturesCommand}"/>
4    </i:EventTrigger>
5  </i:Interaction.Triggers>
```

Code Excerpt 13 – View executing relay command

```
1   public ICommand LoadEditingFeaturesCommand
2   {
3     get
4      {
5        if (_loadEditingFeatures == null)
6        {
7          _loadEditingFeatures = new RelayCommand<object>((_) => LoadEditingFeatures());
8        }
9            return _loadEditingFeatures;
10    }
11  }
```

Code Excerpt 14 – Relay command executing method

4.1.2 Final Solution

On this subchapter the result of the application is presented, thus, the focus of it is on the graphical side of the application.

Along with a brief description of each *graphical interface*, the final implementation status of the use cases exposed on the Analysis section is presented.

Use Cases' Implementation State

With the graphical side of the application tightly coupled to the use cases' implementation state, it is presented the final state of the use cases exposed in the Analysis section.

As seen in Table 2, of all fifteen use cases only one was not implemented, resulting, therefore, on an almost complete implementation of the graphical side of the application.

Table 2 – Use Cases' Implementation State

ID	Use Case	Implementation State
1	The user imports a technical drawing	Implemented
2	The user closes a technical drawing	Implemented
3	The user starts the auto ballooning	Implemented
4	The user drags a shape	Implemented
5	The user rotates a shape	Implemented
6	The user scales a shape	Implemented
7	The user pans and zooms the drawing and the layers	Not Implemented
8	The user deletes a layer	Implemented
9	The user adds a shape	Implemented
10	The user edits a shape	Implemented
11	The user drags an annotation	Implemented
12	The user scales an annotation	Implemented
13	The user adds an annotation	Implemented
14	The user edits an annotation	Implemented
15	The user exports the layers	Implemented

Application Graphical User Interfaces

With simplicity and intuitiveness as major factors to follow in the development of the graphical side, it was decided for the use of just one *interface* for the whole application. This way, any use case can be started from this interface without the need for the users to access any pop-us, sub-windows or hidden menus.

For the theme of the application it was used FARO's WPF theme so that once this application is integrated into FARO CAM2 Software, it follows the same theme.

Figure 14 displays the *interface* before any user action. At this point, the user has access to a menu with four options: Open File (*UC 1*), Close File (*UC 2*), Auto Ballooning (*UC 3*) and Export (*UC 15*). Of which only the first two are unlocked. The menu can also be opened or closed, allowing for the press of its buttons in both states.

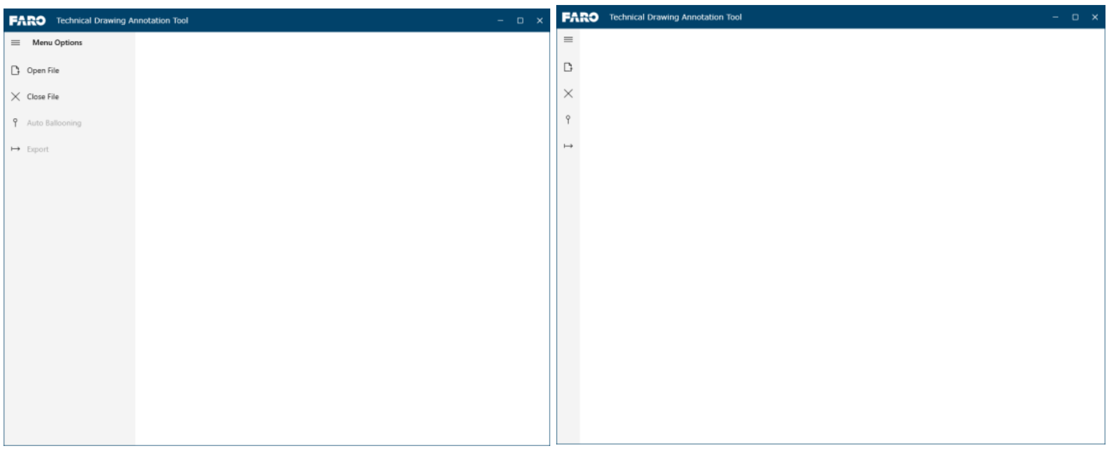
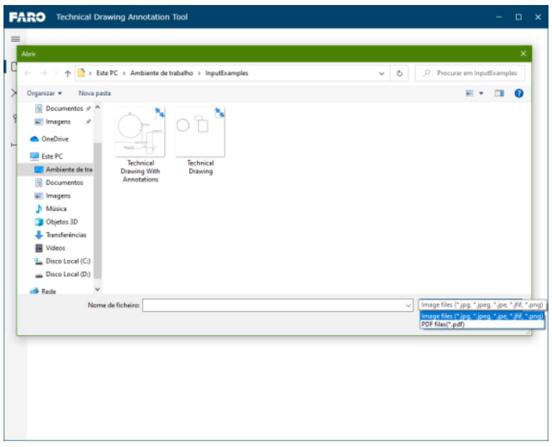


Figure 14 – Application initial interface

Pressing the Open File button, an Open File Dialog opens prompting the user to select a technical drawing. This prompt as seen in Figure 15 allows for the selection of drawings of, both, image and *PDF* formats, which once selected are shown in the application, as seen in Figure 16.


FARO
Technical Drawing Annotation Tool
Abrir
Este PC › Ambiente de trabalho › InputExamples
Procurar em InputExamples
Organizar
Nova pasta
Technical Drawing With Annotations
Technical Drawing
Nome de ficheiro:
Image files (*.jpg, *.jpeg, *.jpe, *.jfif, *.png)
PDF files(*.pdf)

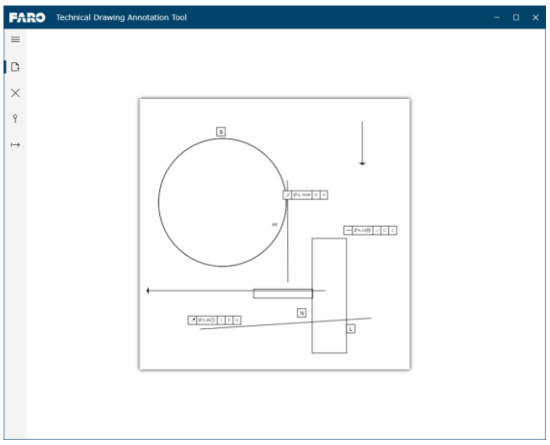


Figure 15 – Importing a technical drawing *Figure 16 – Displaying a technical drawing*

With the technical drawing imported and displayed, the Auto Ballooning (*UC 3*) button is unlocked, therefore, allowing its selection. By selecting it, the image is ballooned and a new menu with the ballooned shapes and annotations appears (Figure 17). The user can open or close each menu item either by clicking on the item name or by double-clicking the shapes or annotations. Each menu item allows for the deletion (*UC 8*) and editing of a shape (*UC 10*) or an annotation (*UC 14*), depending on what is associated with. The deletion (*UC 8*) can also be done by clicking on a shape or annotation and pressing the Delete key.

This menu also allows for the addition of shapes (*UC 9*) and annotations (*UC 13*) through the buttons placed on the bottom of it (Figure 17). The user just needs to select what he wants to add by pressing the corresponding button and then click on the place of the image where he wants it to be added. When adding, and after clicking on the location to add, the user can, by continuing to press the mouse, scale the shape or annotation that was just added.

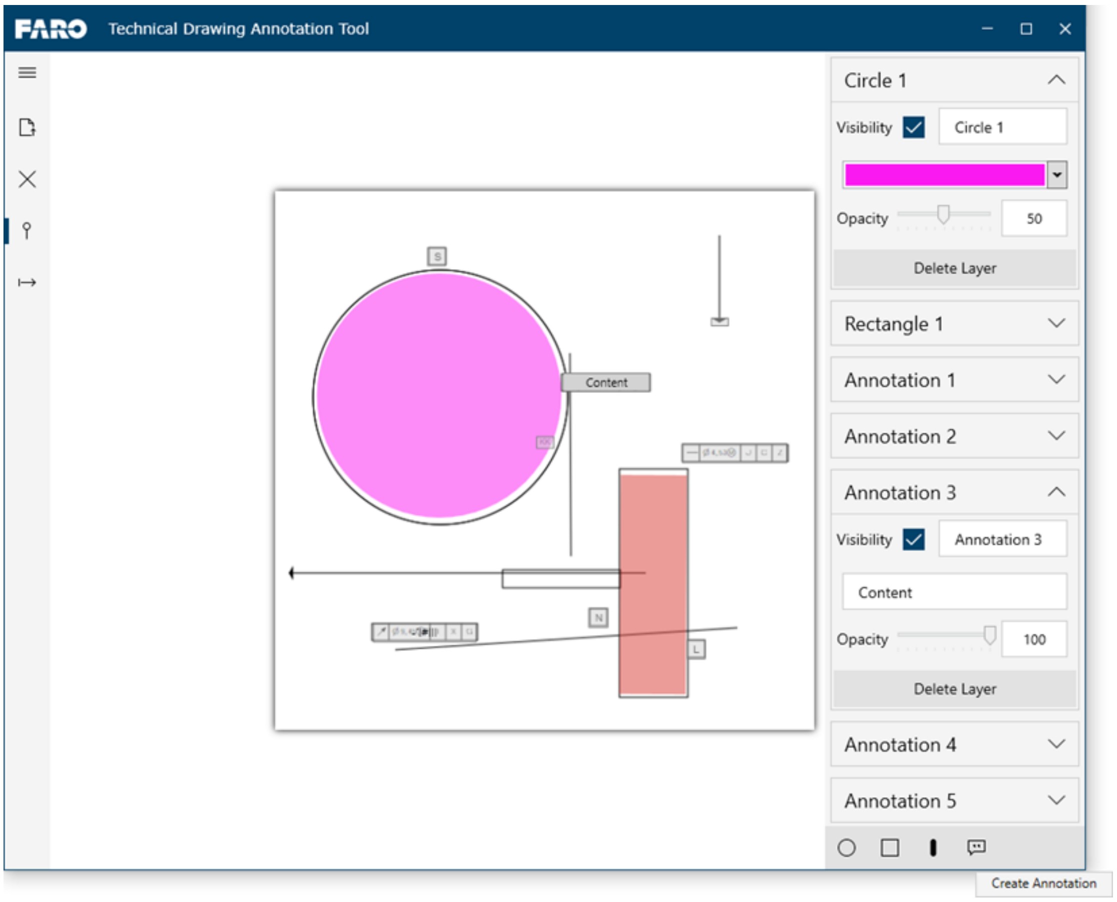


Figure 17 – Application interface after auto ballooning

At this point, besides adding and deleting shapes and annotations, users can also drag (*UC 4 and 11*), rotate (*UC 5)* and scale (*UC 6 and 12*) them. Rectangles and round slots allow for the three features, while circles and annotations do not allow for the rotation. Circles, also, only allow proportional scaling so that they cannot turn into ellipses. Both the scaling and rotation are done through an adorner that is shown around the selected shape or annotation (Figure 18).

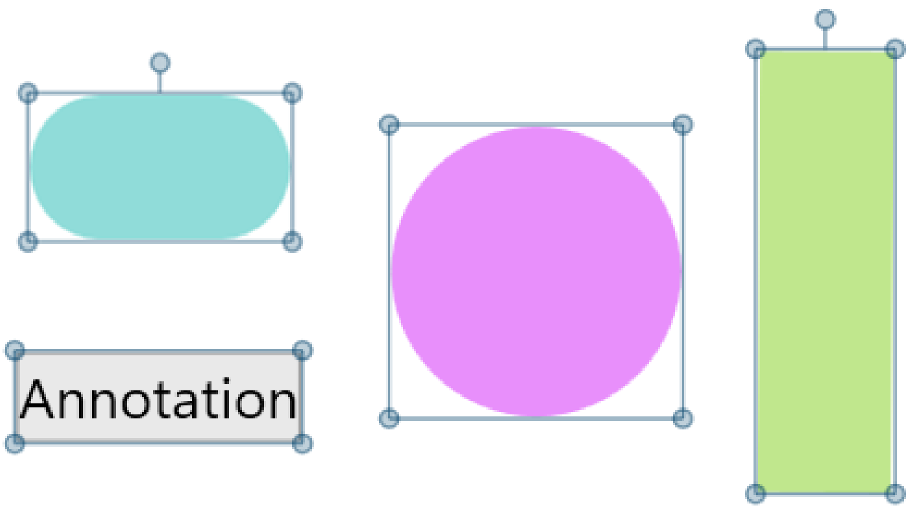


Figure 18 – Shapes and annotations adorners

4.2 Tests

On this subchapter, the tests performed to guarantee the quality of the software developed are presented.

4.2.1 Unit Testing

For the development and execution of the unit tests, it was used the *unit testing frameworks* xUnit and NSubstitute.

xUnit is a free, *open-source*, community focused testing tool [64] that was used for the execution of the tests, being additionally characterized by being of simple integration, stable and allowing the running of tests in parallel [65].

NSubstitute, as xUnit, is also a free and *open-source* testing tool, but focused on providing a friendly approach to *mocking* [66].

Additionally to the use of the mentioned *frameworks*, all *unit tests* followed an AAA (Arrange-Act-Assert) approach, coupled with the use of FluentAssertions, which is a *NuGet package* focused on providing a more natural and readable way of performing the tests assertions [67].

Code Excerpt 15 exposes a *unit test* using all the mentioned packages and the AAA approach, where xUnit is used to run the test (Code Excerpt 15, Line 10), FluentAssertions to perform the test assertion (Code Excerpt 15, Line 26) and NSubstitute to allow the testing of the publishing of an event by *mocking* the Event Aggregator (Code Excerpt 15, Line 27).

```
public class LayerRepositoryTests : BaseContainer
{
  private readonly IEventAggregator eventAggregatorMock;

  public LayerRepositoryTests()
  {
    eventAggregatorMock = Substitute.For<IEventAggregator>();
  }

  [Fact]
  public void LayerRepository_DeleteLayer_Deleted()
  {
    // Arrange
    LayerRepository layerRepository = new LayerRepository(eventAggregatorMock);
    SetupLayerRepository(layerRepository);

    List<Layer> expectedResult = GetLayers();
    Layer layer = expectedResult[2];
    expectedResult.RemoveAt(2);

    // Act
    layerRepository.DeleteLayer(layer);
    List<Layer> result = layerRepository.GetAllLayers();

    // Assert
    expectedResult.Should().BeEquivalentTo(result);
    eventAggregatorMock.Received().PublishOnUIThreadAsync(Arg.Is<LayerDeletedEventModel>(x => x.Index == 2));
  }
}
```

Code Excerpt 15 – Unit test using xUnit and NSubstitute

At last to improve the overall testing of the application it was created two helper classes. One to allow for the using and testing of the *IoC* container and the other to allow for the testing of the property changed notifications.

Code Excerpt 16 exemplifies a *unit test* using both of the mentioned helper classes, with the **BaseContainer** class allowing for the use of the *IoC* container in the tests (Code Excerpt 16, Line 7 and 15) and the **NotifyPropertyChangedTester** allowing for the testing of the property changed notifications (Code Excerpt 16, Line 18 and 22) [68].

```
public class AnnotationViewModelTests : BaseContainer
{
  public ViewModelFactoryRepository ViewModelFactoryRepository;

  public AnnotationViewModelTests()
  {
    ViewModelFactoryRepository = AppContainer.Resolve<ViewModelFactoryRepository>();
  }

  [Fact]
  public void AnnotationViewModel_Annotation_Opacity_PropertyChanged_Notification()
  {
    // Arrange
    Annotation annotation = new Annotation(0, new Rectangle(0, Colors.Red, 50.0, new Point(0.0, 0.0), new Point(50.0, 0.0), new Point(50.0, 50.0), new Point(0.0, 50.0)), "Annotation");
    AnnotationViewModel annotationViewModel = ViewModelFactoryRepository.AnnotationViewModelFactory.Create(annotation, false);

    // Act
    NotifyPropertyChangedTester notifyPropertyChangedTester = new NotifyPropertyChangedTester(annotationViewModel);
    annotation.AnnotationRectangle.Opacity = 40.0;

    // Assert
    notifyPropertyChangedTester.AssertPropertyChangedNotification(0, "Opacity");
  }
}
```

Code Excerpt 16 – Unit test using both helper classes

Also, besides the mentioned classes it was necessary to use the *NuGet package* Xunit.StaFact for allowing the execution of all tests that use visual elements, as seen in the Code Excerpt 17, since these tests only work if run on *STA threads*.

```
[StaFact]
public void IsConstrainedFalse_DragOutsideParentBounds_LeavesParentBounds()
{
  // Arrange
  Rectangle rect = CreateRectangleInGrid();
  DragElementBehavior behavior = CreateAndAttachDragElementBehavior(rect);
  // Act
  this.PerformSingleDrag(behavior, new Point(5, 5), new Point(101, 101));
  // Assert
  this.VerifyOffset(rect.RenderTransform, 96, 96);
}
```

Code Excerpt 17 – Unit testing using Xunit.StaFact

Besides the mentioned development options and technologies, it was also decided for the implementation of all tests in a separate project that references the application. This way, by just changing the project reference, multiple versions of the application can be tested using the same tests.

Adding to this, by providing the tests separately the application can be deployed without them, therefore, having a smaller size.

4.3 Solution evaluation

Following a Scrum approach throughout the development of the system, there was a constant evaluation of the solution by both the Product Owner and all the organization members present in the sprint reviews.

This evaluation came in form of feedback, therefore, not following any official software evaluation guides, but still allowing for the information transmitted to enable the alignment of the solution with what was expected from the Product Owner.

Throughout the development of the project several evaluations were done, mainly during sprint reviews, allowing, therefore, for a constant shape of the solution.

With the end of the development there was one last evaluation, with the presented solution having been considered, by both the Product Owner and the author, as successful, despite some of the small limitations it presents. This result come from a comparison of the solution with the requested requirements.

The final solution can, therefore, be considered fully functional, having passed all the usability, reliability and performance constraints set by the Product Owner and presented on the Non-functional requirements section.

5 Conclusions

On this chapter, the conclusions reached through the development process are presented, demonstrating the comparison between the obtained results and the expected ones. It is also demonstrated the analysis of the work done, where the limitations of the work are shown along with possible future development paths. This chapter ends with an opinion from the author about the work developed.

5.1 Objectives achieved

On this subchapter, the survey of the accomplished objectives is done. The main objectives are defined in Section 3.1.3 of this document, as well as all the features that the final solution should offer in section 3.1.2, where the identification of use cases is made. The successful implementation of these is also presented in section 4.1.2.

There were fifteen (15) use cases to be completed, with fourteen (14) completed and one (1) to be completed, with its incompletion manly due to a poor execution planning.

Despite the non-completed UC, the overall strict objectives defined in the Objectives section of the Introduction can be considered complete, with the solution providing an intuitive and easy process for the gathering of the PMI data of technical drawings. Having said that, the layout and usability of the application are not yet at the level completely desired by both the user and the Product Owner.

5.2 Limitations and future work

Although the final solution is functional and already allows for the use of a very interesting set of features, it still has some limitations.

Being the main one the import of technical drawings in PDF format. The necessity of this support is imperative with most technical drawings being in PDF, rather than image, format. Nonetheless, due to both the WPF application and the backend only supporting image formats it is crucial for the application to be able to convert the imported PDFs into images. However, because of the lack of good and free C# solutions, this conversion was implemented through the execution of a python executable, which not only is very slow, as it also requires a good amount of processing power, therefore, contradicting both the optimization requirement and the objective of reducing the time for the gathering of the PMI data.

Besides this main limitation, all other limitations are visual related and happen due to bugs, that were not able to be fixed before the end of the internship in, which, the project was developed. Although these bugs are noticeable to the user and slightly lower the usability of the application, none prevent any action from being accomplished.

Apart from improving and fixing the mentioned limitations, regarding future work, the application should also add support for the gathering and editing of the technical drawings' connections *PMI* data, which, are the lines, present in the drawings, that associate the annotations to the shapes. But, for this, it is also necessary for the support of these connections from both the backend and the deep learning algorithms.

Finally, to further improve the intuitiveness and usability, the application should also provide more shortcut keyboards for the executing of certain actions, like for example the selection of the initial menu buttons.

5.3 Final Assessment

In retrospect, as the author, I consider myself satisfied with the developed project, however, fully aware that the application is still far from ready for the market.

Antares objective was for the elaboration of a solution that respected all their requests, and, in the end, with the solution developed, I can say that apart from one use case that was not implemented, those requests were met. I, therefore, consider that my contribution to the company was positive.

With the development of this project, I've come to a quick realization that good planning is key for a successful implementation, but that no matter how much planning is done there is always a chance that changes happen, therefore, the implementation should follow the highest standards possible, through the use of good practices, like *SOLID principles*, software patterns and testing, to allow for easier code maintainability.

That said, and given the elaboration of this document as finished, I want to emphasize the satisfaction and growth, both personal and technical, that this whole process as provided. As well as give a word of support for how the Informatics Engineering Degree (LEI) of ISEP is structured. It was thanks to the knowledge acquired through it that I was able to complete this project, allowing me to quickly grasp the used technologies, even though I had never used them before.

Lastly, I want to leave a word of appreciation to, both, the professor Paulo Proença and the whole Antares team for all the availability and help presented throughout this project development.

References

[1] Technical drawing. (2020, April 25). Retrieved April 2, 2020, from https://en.wikipedia.org/wiki/Technical_drawing#Engineering

[2] Scrum (software development). (2020, April 25). Retrieved April 26, 2020, from https://en.wikipedia.org/wiki/Scrum_(software_development)

[3] What is Scrum? (n.d.). Retrieved April 26, 2020, from https://www.scrum.org/resources/what-is-scrum

[4] Inspection Manager. (2020, May 6). Retrieved June 16, 2020, from https://www.highqa.com/inspection-manager/

[5] High QA introduces Inspection Manager software version 5.0. (2020, February 24). Retrieved June 16, 2020, from https://www.thefabricator.com/thefabricator/product/cadcamsoftware/high-qa-introduces-inspection-manager-software-version-50

[6] Quality Engineering Tool Software, Easy to Use: DISCUS Software. (2020, May 12). Retrieved June 16, 2020, from https://www.discussoftware.com/

[7] discussoftware (2019, October 31). DISCUS Software Auto Ballooning [Video]. YouTube. Retrieved June 16, 2020, from https://www.youtube.com/watch?v=Z44EkqLcNMU

[8] DISCUS OCR. (2019, July 23). Retrieved June 16, 2020, from https://www.discussoftware.com/fai-optimization/products/desktop/ocr/

[9] DISCUS Planner. (2019, November 6). Retrieved June 16, 2020, from https://www.discussoftware.com/fai-optimization/products/desktop/planner/

[10] 3D Data Management Use Native 3D CAD model (CATIA, Pro/E, NX). (2019, July 23). Retrieved June 16, 2020, from https://www.discussoftware.com/fai-optimization/products/desktop/3d/

[11] InspectionXpert in 2020 - Reviews, Features, Pricing, Comparison. (2020, January 11). Retrieved June 16, 2020, from https://www.predictiveanalyticstoday.com/inspectionxpert/

[12] Kasilingam, S. (2018, November 7). InspectionXpert 2.0 SP7. Retrieved June 16, 2020, from https://www.inspectionxpert.com/release-notes/inspectionxpert-sp7

[13] InspectionXpert Corporation. (n.d.). InspectionXpert Ballooning Software. Retrieved June 16, 2020, from https://www.inspectionxpert.com/

[14] Net-Inspect. (n.d.). Supply Chain Management and Quality Management Software: Net-Inspect. Retrieved June 16, 2020, from https://www.net-inspect.com/

[15] FARO® CAM2® Software: FARO Technologies. (n.d.). Retrieved May 7, 2020, from https://www.faro.com/products/3d-manufacturing/cam2-software/

[16] Electron (software framework). (2020, May 22). Retrieved May 26, 2020, from https://en.wikipedia.org/wiki/Electron_(software_framework)

[17] Shpilt, M. (2019, January 15). 9 Must Decisions in Desktop Application Development for Windows. Retrieved May 25, 2020, from https://michaelscodingspot.com/9-must-decisions-in-desktop-application-development-for-windows/

[18] Electron Support. (n.d.). Retrieved May 26, 2020, from https://www.electronjs.org/docs/tutorial/support

[19] Fernström, M. (2018, December 04). What are the advantages and disadvantages of Electron to build cross platform desktop applications?. Retrieved May 26, 2020, from https://www.quora.com/What-are-the-advantages-and-disadvantages-of-Electron-to-build-cross-platform-desktop-applications

[20] About Qt. (2019, April 26). Retrieved May 27, 2020, from https://wiki.qt.io/About_Qt

[21] Download Qt Open Source. (n.d.). Retrieved May 27, 2020, from https://www.qt.io/download-open-source

[22] Company, T. Q. (n.d.). Cross-platform software development for embedded & desktop. Retrieved May 27, 2020, from https://www.qt.io/

[23] Blanchette, Jasmin; Summerfield, Mark (June 2006). "A Brief History of Qt". C++ GUI Programming with Qt 4 (1st ed.). Prentice-Hall. pp. xv–xvii. Retrieved May 27, 2020.

[24] Qt (software). (2020, May 27). Retrieved May 27, 2020, from https://en.wikipedia.org/wiki/Qt_(software)

[25] Qt Project Open Governance. (n.d.). Retrieved May 27, 2020, from https://wiki.qt.io/Qt_Project_Open_Governance

[26] QtReleasing. (n.d.). Retrieved May 27, 2020, from https://wiki.qt.io/QtReleasing

[27] Company, T. Q. (n.d.). Supported Platforms & Languages. Retrieved May 27, 2020, from https://www.qt.io/product/supported-platforms-languages

[28] Supported Platforms: Qt 5.15. (n.d.). Retrieved May 27, 2020, from https://doc.qt.io/qt-5/supported-platforms.html

[29] Wezorek, J. (2013, November 30). What are the Pros and Cons of using QT framework for cross platform programming (Win/Mac)? Retrieved May 27, 2020, from https://www.quora.com/What-are-the-Pros-and-Cons-of-using-QT-framework-for-cross-platform-programming-Win-Mac

[30] ONeal, B. (2012, November 20). Why aren't more desktop apps written with Qt? Retrieved May 27, 2020, from https://softwareengineering.stackexchange.com/questions/88685/why-arent-more-desktop-apps-written-with-qt

[31] JavaFX. (2020, May 31). Retrieved May 31, 2020, from https://en.wikipedia.org/wiki/JavaFX

[32] Swing (Java). (2020, May 11). Retrieved May 31, 2020, from https://en.wikipedia.org/wiki/Swing_(Java)

[33] Hodkinson, T. (2018, March 19). JavaFX and the Future of Java Client Technologies. Retrieved June 1, 2020, from https://www.infoq.com/news/2018/03/JavaFXRemovedFromJDK/

[34] Openjdk. (n.d.). openjdk/jfx. Retrieved June 1, 2020, from https://github.com/openjdk/jfx/blob/jfx14/doc-files/release-notes-14.md

[35] FXML. (2020, March 5). Retrieved June 1, 2020, from https://en.wikipedia.org/wiki/FXML

[36] JavaFX Scene Builder A Visual Layout Tool for JavaFX Applications (n.d.). Retrieved June 1, 2020, from https://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html

[37] Tyson, M. (2020, January 17). What is the JVM? Introducing the Java Virtual Machine. Retrieved June 1, 2020, from https://www.javaworld.com/article/3272244/what-is-the-jvm-introducing-the-java-virtual-machine.html

[38] (n.d.). Retrieved June 1, 2020, from https://geeks-world.github.io/articles/464327/index.html

[39] Lotfaliei, Matin. (2016, May 21). WPF vs Qt Widget in CPU performance and UI framerate [Video]. YouTube. Retrieved June 1, 2020, from https://www.youtube.com/watch?v=8O-0k4M1LKs8

[40] Windows Presentation Foundation. (2020, May 3). Retrieved June 2, 2020, from https://en.wikipedia.org/wiki/Windows_Presentation_Foundation

[41] Protalinski, E. (2019, September 23). Microsoft releases .NET Core 3.0 with support for WPF and Windows Forms. Retrieved June 2, 2020, from https://venturebeat.com/2019/09/23/microsoft-releases-net-core-3-0-with-support-for-wpf-and-windows-forms/

[42] WPF vs. WinForms. (n.d.). Retrieved June 2, 2020, from https://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/

[43] Dotnet. (2020, June 7). dotnet/wpf. Retrieved June 7, 2020, from https://github.com/dotnet/wpf

[44] Thraka. (n.d.). XAML overview - WPF. Retrieved June 7, 2020, from https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml

[45] Cocoa (API). (2020, March 7). Retrieved June 10, 2020, from https://en.wikipedia.org/wiki/Cocoa_(API)

[46] Abstract Window Toolkit. (2019, December 16). Retrieved June 10, 2020, from https://en.wikipedia.org/wiki/Abstract_Window_Toolkit

[47] Swing (Java). (2020, May 11). Retrieved June 10, 2020, from https://en.wikipedia.org/wiki/Swing_(Java)

[48] Standard Widget Toolkit. (2020, May 11). Retrieved June 10, 2020, from https://en.wikipedia.org/wiki/Standard_Widget_Toolkit

[49] Universal Windows Platform. (2020, May 21). Retrieved June 10, 2020, from https://en.wikipedia.org/wiki/Universal_Windows_Platform

[50] Windows Forms. (2020, May 9). Retrieved June 10, 2020, from https://en.wikipedia.org/wiki/Windows_Forms

[51] Dotnet. (2020, June 5). dotnet/winforms. Retrieved June 10, 2020, from https://github.com/dotnet/winforms

[52] Mcleanbyron. (n.d.). Choose your app platform. Retrieved June 10, 2020, from https://docs.microsoft.com/en-us/windows/apps/desktop/choose-your-platform

[53] Xamarin. (n.d.). xamarin/Xamarin.Forms. Retrieved June 16, 2020, from https://github.com/xamarin/Xamarin.Forms/wiki/Platform-Support

[54] FURPS. (2020, April 04). Retrieved June 18, 2020, from https://en.wikipedia.org/wiki/FURPS

[55] Coepd. (2014, August 05). What is FURPS+? Retrieved June 18, 2020, from https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/

[56] Campos, P. F. (2009, February 03). FURPS. Retrieved June 18, 2020, from https://qualidadebr.wordpress.com/2008/07/10/furps/

[57] Dotnet. (n.d.). Dotnet/core. Retrieved June 18, 2020, from https://github.com/dotnet/core/blob/master/release-notes/3.0/3.0-supported-os.md

[58] Actor (UML). (2019, November 14). Retrieved June 18, 2020, from https://en.wikipedia.org/wiki/Actor_(UML)

[59] Model-View-ViewModel (MVVM) Explained. (2018, May 18). Retrieved June 22, 2020, from https://www.wintellect.com/model-view-viewmodel-mvvm-explained/

[60] Laubheimer, P. (2020, February 23). Drag–and–Drop: How to Design for Ease of Use. Retrieved June 22, 2020, from https://www.nngroup.com/articles/drag-drop/

[61] Caliburn.Micro · 'Xaml made easy' · Caliburn.Micro. (n.d.). Retrieved June 23, 2020, from https://caliburnmicro.com/

[62] Basic Configuration, Actions and Conventions. (n.d.). Retrieved June 23, 2020, from https://caliburnmicro.com/documentation/configuration

[63] Castleproject. (2019, November 27). Castleproject/Windsor. Retrieved June 23, 2020, from https://github.com/castleproject/Windsor/blob/master/docs/README.md

[64] About xUnit.net. (n.d.). Retrieved June 25, 2020, from https://xunit.net/

[65] Running Tests in Parallel. (n.d.). Retrieved June 25, 2020, from https://xunit.net/docs/running-tests-in-parallel

[66] Nsubstitute. (n.d.). Nsubstitute/NSubstitute. Retrieved June 25, 2020, from https://github.com/nsubstitute/NSubstitute

[67] Fluent Assertions. (2016, March 23). Retrieved June 25, 2020, from https://fluentassertions.com/

[68] Day, B. (2010, August 24). An easier way to unit test INotifyPropertyChanged in Silverlight/WPF. Retrieved June 25, 2020, from https://www.benday.com/2010/08/24/an-easier-way-to-unit-test-inotifypropertychanged-in-silverlightwpf/

Attachment A – UC 3 Sequence Diagram

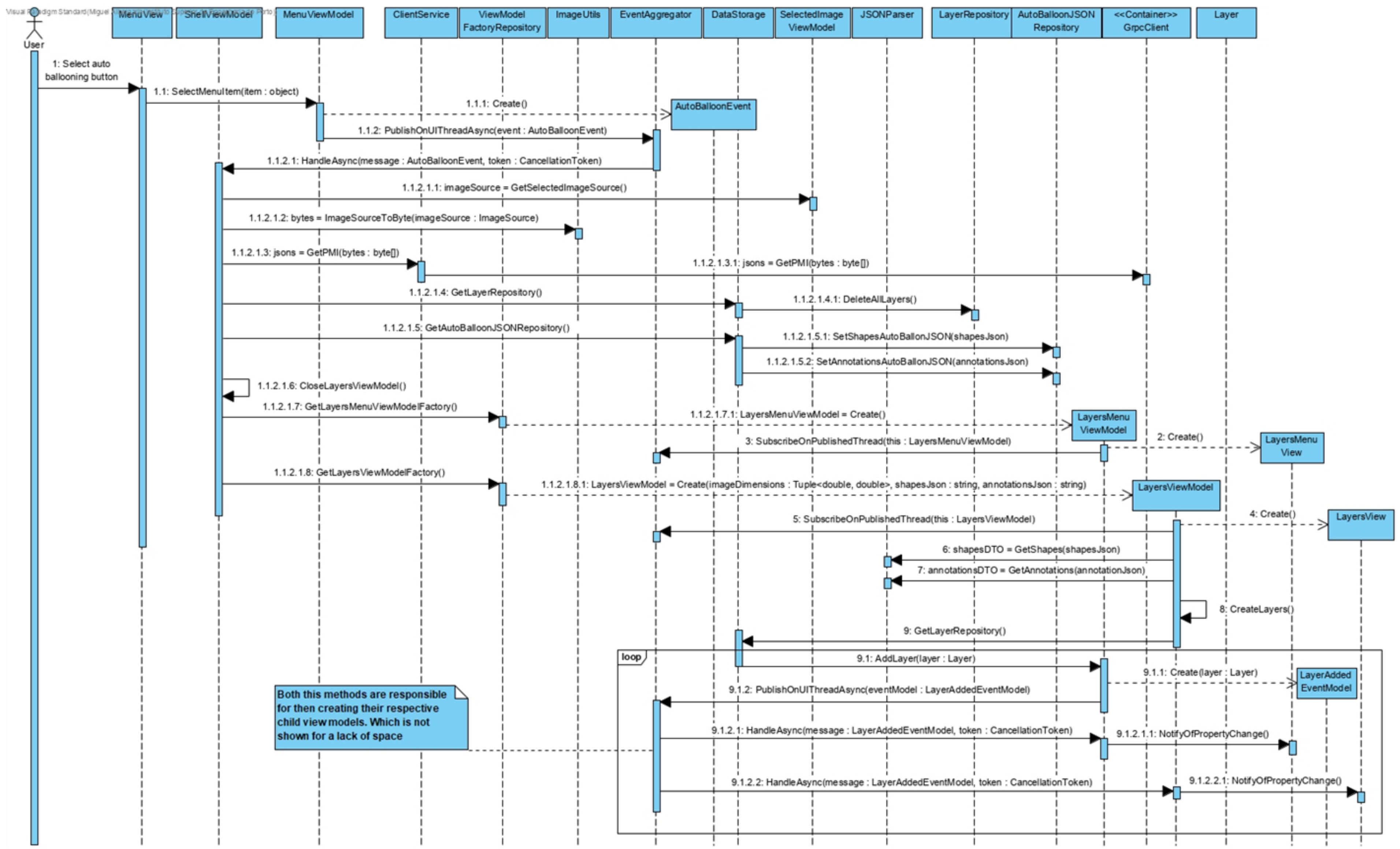

User
MenuView
ShellViewModel
MenuViewModel
ClientService
ViewModel FactoryRepository
ImageUtils
EventAggregator
DataStorage
SelectedImage ViewModel
JSONParser
LayerRepository
AutoBalloonJSON Repository
<<Container>> GrpcClient
Layer
1: Select auto ballooning button
1.1: SelectMenuItem(item : object)
1.1.1: Create()
AutoBalloonEvent
1.1.2: PublishOnUIThreadAsync(event : AutoBalloonEvent)
1.1.2.1: HandleAsync(message : AutoBalloonEvent, token : CancellationToken)
1.1.2.1.1: imageSource = GetSelectedImageSource()
1.1.2.1.2: bytes = ImageSourceToByte(imageSource : ImageSource)
1.1.2.1.3: jsons = GetPMI(bytes : byte[])
1.1.2.1.3.1: jsons = GetPMI(bytes : byte[])
1.1.2.1.4: GetLayerRepository()
1.1.2.1.4.1: DeleteAllLayers()
1.1.2.1.5: GetAutoBalloonJSONRepository()
1.1.2.1.5.1: SetShapesAutoBallonJSON(shapesJson)
1.1.2.1.5.2: SetAnnotationsAutoBallonJSON(annotationsJson)
1.1.2.1.6: CloseLayersViewModel()
1.1.2.1.7: GetLayersMenuViewModelFactory()
1.1.2.1.7.1: LayersMenuViewModel = Create()
LayersMenu ViewModel
2: Create()
LayersMenu View
3: SubscribeOnPublishedThread(this : LayersMenuViewModel)
1.1.2.1.8: GetLayersViewModelFactory()
1.1.2.1.8.1: LayersViewModel = Create(imageDimensions : Tuple<double, double>, shapesJson : string, annotationsJson : string)
LayersViewModel
4: Create()
LayersView
5: SubscribeOnPublishedThread(this : LayersViewModel)
6: shapesDTO = GetShapes(shapesJson)
7: annotationsDTO = GetAnnotations(annotationJson)
8: CreateLayers()
9: GetLayerRepository()
loop
9.1: AddLayer(layer : Layer)
9.1.1: Create(layer : Layer)
LayerAdded EventModel
9.1.2: PublishOnUIThreadAsync(eventModel : LayerAddedEventModel)
Both this methods are responsible for then creating their respective child view models. Which is not shown for a lack of space
9.1.2.1: HandleAsync(message : LayerAddedEventModel, token : CancellationToken)
9.1.2.1.1: NotifyOfPropertyChange()
9.1.2.2: HandleAsync(message : LayerAddedEventModel, token : CancellationToken)
9.1.2.2.1: NotifyOfPropertyChange()

Figure 19 - The user starts the auto ballooning (Sequence Diagram)