Converting an existing .NET Framework ground control software into a cross-platform application

Konvertera en existerande .NET Framework markstationsmjukvara till en multiplattformsapplikation

Erik Boman

Supervisor : Mikael Asplund
Examiner : Simin Nadjm-Tehrani

External supervisor : Paul Holmstedt
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera åktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydda mot att dokumentet ändras eller presenteras i sådan form eller i sådan sammanhang som är krävande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida [http://www.ep.liu.se/].

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/her own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: [http://www.ep.liu.se/].

© Erik Boman
Abstract

Unmanned aerial vehicles can be used in many different situations such as, for example, monitoring the growth of crops or for surveillance of a private property. Operating the unmanned aerial vehicle is usually done using some kind of ground control station. This thesis examines the possibilities of creating ground control stations working on several different platforms using the cross platform development frameworks Xamarin, Universal Windows Platform and Mono. This is done by creating and comparing three prototype applications regarding functional requirements, code reuse and resource usage. It is shown that none of the cross platform frameworks can fulfill all of the initial requirements on a ground control station. However, for the case studied in this thesis, Xamarin is demonstrated to be the most suitable cross platform framework of the three since it provides the same functionality as UWP for Windows devices while also enabling development for both Android and iOS devices.
Acknowledgments

First, I would like to thank my examiner Simin Nadjm-Tehrani and my supervisor Mikael Asplund for your high engagement in this thesis and in the writing of this thesis report. Without your valuable comments on the structure and content of this thesis, it is likely that the end result would have been significantly different.

Another person I would like to thank is Paul Holmstedt at UAS Europe AB. Thank you for a great thesis subject and all the help regarding details about SkyView GCS.

I would also like to thank the rest of the employees at UAS Europe AB and Instrument Control Sweden AB for a nice and welcoming working environment.

At last, I would like to thank my family for all your support and encouragement.

Linköping, December 2017
Erik Boman
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview of an Unmanned Aerial System</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Motivation</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Creating applications for multiple platforms</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Problem statement</td>
<td>4</td>
</tr>
<tr>
<td>1.5 Method for this thesis</td>
<td>5</td>
</tr>
<tr>
<td>1.6 Delimitations</td>
<td>5</td>
</tr>
<tr>
<td>1.7 Thesis overview</td>
<td>5</td>
</tr>
<tr>
<td>2 Background</td>
<td>7</td>
</tr>
<tr>
<td>2.1 UAS Europe AB</td>
<td>7</td>
</tr>
<tr>
<td>2.2 SkyView GCS</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Agri/Spy Owl</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Functional requirements</td>
<td>10</td>
</tr>
<tr>
<td>3 Multi-platform development frameworks</td>
<td>11</td>
</tr>
<tr>
<td>3.1 The Common Language Infrastructure</td>
<td>12</td>
</tr>
<tr>
<td>3.2 Common Language Infrastructure implementations</td>
<td>13</td>
</tr>
<tr>
<td>3.3 .NET Standard</td>
<td>16</td>
</tr>
<tr>
<td>3.4 Related work</td>
<td>17</td>
</tr>
<tr>
<td>4 Development and evaluation methods</td>
<td>19</td>
</tr>
<tr>
<td>4.1 UWP application</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Xamarin application</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Mono application</td>
<td>23</td>
</tr>
<tr>
<td>4.4 Testing functional requirements</td>
<td>24</td>
</tr>
<tr>
<td>4.5 Evaluating code reuse across platforms</td>
<td>25</td>
</tr>
<tr>
<td>4.6 Evaluating code reuse from SkyView GCS</td>
<td>26</td>
</tr>
<tr>
<td>4.7 Evaluating resource usage</td>
<td>27</td>
</tr>
<tr>
<td>5 Results</td>
<td>29</td>
</tr>
<tr>
<td>5.1 Prototype implementations</td>
<td>30</td>
</tr>
<tr>
<td>5.2 Functional requirements</td>
<td>30</td>
</tr>
</tbody>
</table>
6 Discussion
  6.1 Method ........................................ 43
  6.2 Results ...................................... 44
  6.3 The work in a wider context .................. 47

7 Conclusion
  7.1 Answers to the questions in the problem statement ..... 48
  7.2 Future work .................................... 49

Bibliography ............................................. 51
List of Figures

1.1 Overview of a typical UAS ......................................................... 2
2.1 Overview of the components in SkyView GCS ............................... 8
2.2 The SkyView GCS software GUI ............................................... 9
2.3 Image of Agri Owl 200 .............................................................. 9
3.1 An overview of the CLI, the CLI implementations and the applications related to this thesis. The image is adapted from an image created by Microsoft. .... 12
3.2 Devices that can run applications written for UWP .............................. 14
3.3 Overview of how code is structured in Xamarin ............................. 15
3.4 The .NET Standard’s relation to the CLI and the CLI implementations .... 16
4.1 The Xamarin prototype application is structured into different projects .......... 22
4.2 Time-line showing when each evaluation-event is planned to occur ............ 27
4.3 The locations used for the resource evaluation. Linköping, Louiseville, Siapa, Bass strait and Karthaus from top left to bottom right. All the images are from the UWP application using Bing Maps. ........................................ 28
5.1 The prototypes created using Mono (top left), UWP (top right) and Xamarin (bottom). The applications are running on Linux, the Windows 10 PC and the Android mobile respectively .................................................. 30
5.2 The CPU usage for the UWP application ......................................... 35
5.3 The memory usage for the UWP application ..................................... 36
5.4 The CPU usage for the Xamarin application ..................................... 37
5.5 The memory usage for the Xamarin application .................................. 38
5.6 The CPU usage for the Mono application ......................................... 39
5.7 The memory usage for the Mono application. In the diagram Windows 10 PC is represented by the blue line and Linux is represented by the yellow line .......... 40
5.8 The average CPU usage over the course of the evaluation for each method and platform evaluated ....................................................... 41
5.9 The maximum memory usage for each method and platform evaluated .......... 42
List of Tables

5.1 Comparison of the different framework and platform combinations covering functional requirements. Acronyms used: W=Windows 10, WM=Windows 10 Mobile, A=Android, L=Linux .......................................................... 31

5.2 The differences in functionality between the platforms in the Xamarin application. Acronyms used: W=Windows 10, WM=Windows 10 Mobile, A=Android. .......... 32

5.3 The estimated amount of code that can be reused across the platforms. Acronyms used: W=Windows 10, WM=Windows 10 Mobile, A=Android, L=Linux . . . . . 33

5.4 The estimated amount of code that can be reused from the original Skyview GCS application. ................................................................. 34
The usage of Unmanned Aerial Vehicles (UAVs) has increased in the last couple of years. UAVs have several advantages over manned vehicles such as decreased weight, higher safety for the operators and lower costs. For these reasons, UAVs can replace manned vehicles in several situations. With the increase in UAV usage, the number of companies involved in building UAVs has also increased leading to higher competition and quickly improving UAVs.

To monitor and control a UAV, some kind of Ground Control Station (GCS), with software specifically developed for monitoring and controlling UAVs is used. The GCS could be anything from a smart phone to a large computer with antennas for communication and special buttons and joysticks for controlling the UAV. The whole system, (UAV, datalink, GCS hardware and software), is referred to as an Unmanned Aerial System (UAS) in the rest of this thesis.

The GCS software is, in most cases, developed for a specific family of GCS hardware. For example a GCS software developed for Windows 7 only works on Windows PCs with operating system Windows 7 or later and a software developed for Android only works on Android smartphones and tablets. However, there are some alternatives that allow the software to run on multiple platforms with different operating systems, reaching more customers without having to recreate the software for each platform. It is some of those alternatives that this thesis investigates.

The work in this thesis is related to the GCS software as well as some of the GCS hardware and has been conducted at a company called UAS Europe AB which is working with UAVs and ground control stations. UAS Europe AB already have a working GCS solution called SkyView GCS developed using the .NET Framework, which contains a set of components that can help the developer write programs. However, using the .NET Framework limits the program to only Windows PCs. This means that SkyView GCS can not be used with, for example, a smartphone or a tablet.
The rest of this chapter is structured in the following way. Section 1.1 contains a short overview of the typical components in a UAS and what parts of a UAS this thesis is related to. In Section 1.2 the motivation behind the thesis is described and in Section 1.3 the cross platform frameworks that was studied in this thesis are presented. In Section 1.4 the problem statement related to this thesis is presented and discussed. After that, in Section 1.5 an overview of how the work was conducted is presented and in Section 1.6 the boundaries of the thesis are explained and some things that are not a part of the thesis are brought up. Lastly, in Section 1.7 an overview and introduction to the content of the different chapters in this thesis is given.

1.1 Overview of an Unmanned Aerial System

A UAS contains several different parts. Typically at least a UAV, a GCS and some kind of communication between them (see Figure 1.1). This section gives an overview of the most important parts and explains which of these parts this thesis will deal with.

1.1.1 Unmanned Aerial Vehicle

The UAV is typically a small to medium sized aircraft or rotorcraft. A UAV can include a wide range of components. The most common are listed and explained here.

- **Power supply and motors**
  Some sort of power supply and motor are needed for the UAV to fly. Most UAVs use batteries as their power supply and electric motors for propulsion, but in some cases liquid fuel and a combustion engine is used instead.

- **Flight controller**
  The flight controller can be more or less complex. Simple flight controllers might just translate steering signals to aileron, elevator, rudder or engine changes while complex flight controllers might be able to fly and even land the UAV by itself.

![Figure 1.1: Overview of a typical UAS.](image)
• Sensors
A UAV typically includes various sensors such as gyroscopes, lasers for measuring its surroundings and cameras for documenting the flight. Some of the sensors might be used just by the flight controller while others are used only by the person operating the UAV.

1.1.2 Communication hardware
To communicate with the UAV some kind of data link is needed. This can be just an ordinary WiFi connection immediately between the GCS and the UAV. However, for communication over longer distances, typically some kind of radio transmitter on the ground and a receiver in the UAV is used. In several cases it is necessary to be able to send data from the UAV to the GCS as well.

1.1.3 Ground Control Station
The main objective of the GCS is to give the operator information about the UAV as well as allowing the operator to steer the UAV. The GCS can be anything from a hand held radio controller to a computer system running complex software and using various screens and displays for showing data to the operator. The data can include anything from the position of the UAV to a live video stream from the UAV camera. More complex GCS software typically include a map that shows the UAVs position in real time. The GCS software is the part of the UAS this thesis will focus on.

1.2 Motivation
Not every GCS hardware fits every situation. In some cases a large desktop is needed due to large computational requirements while in other cases a small hand held device is better suited since it is easy to use and transport. Because of the large variety of different possible GCS hardwares, each with its own operating system, there is a need to develop the software of the GCS so that it can be used across as wide of a range of devices as possible.

Traditionally in software development, one software has had to be created for each family of devices. For example, one application written in C# using the .NET Framework for Windows computers, one written in C or C++ for Linux computers and one written in Java for Android smartphones and tablets. This leads to multiple different code bases that can not be easily shared between the platforms. This, in turn, leads to difficulties in maintaining and further developing the software. On the other hand, if a single code base can be used across multiple platforms, the software becomes easier to maintain and further develop, saving both time and money. It also makes it easier to create applications for different platforms with a uniform user interface, making the learning process easier for the user when switching from one device to another.
1.3 Creating applications for multiple platforms

There are several different approaches on how to create applications for multiple platforms. In the beginning of this thesis UAS Europe AB suggested three different frameworks to look closer at. They are introduced in this section.

- **Universal Windows Platform (UWP)**
  UWP is a platform used to develop applications for any device running a Windows 10 based operating system. This includes normal Windows 10 PCs, Windows 10 Mobiles, Windows 10 IoT devices, Xbox One and HoloLenses.

- **Xamarin**
  Xamarin provides common libraries and tools for creating applications aimed at various mobile devices such as Android and iOS smartphones and tablets as well as Windows 10 and macOS devices. When creating a Xamarin application, UWP is used in the background to allow for the Windows 10 support.

- **Mono**
  Mono is an open source project that include libraries and tools that can be used across multiple different platforms such as various Windows, macOS and Linux versions, including Ubuntu, Debian, CentOS and Raspbian.

These frameworks provide different functionalities which is relevant to how well-suited they are as frameworks for creating a GCS software. UWP, Xamarin and Mono will be further explained in Chapter 3.

1.4 Problem statement

In this thesis the following four questions are studied and answered.

1. **To what extent does UWP, Xamarin and Mono support the functionality needed for creating a cross platform Ground Control System software?**
   A Ground Control System software has certain characteristics which demands some specific functionality to be available. For example, it is necessary to be able to show a map with custom icons on it, save files of arbitrary file format and use the application in offline mode. Investigating how well UWP, Xamarin and Mono support these functionalities would not only be beneficial for UAS Europe AB but also for other projects with similar types of applications.

2. **In what ways does a Ground Control Software developed using UWP, Xamarin and Mono differ regarding CPU and memory usage?**
   The resource demand of an application limits the amount of devices the application can run on. Since the whole point of developing an application using UWP, Xamarin or Mono is to make it accessible to a larger number of devices, it makes sense to investigate if the applications developed will differ regarding CPU and memory usage.

3. **Having a C# .NET Framework application, are there any typical segments of code that has to be rewritten in order to work on the different operating systems, using the different frameworks?**
   Considering SkyView GCS, as a reference case, identifying the typical segments of code that have to be rewritten, and quantifying the extent of code that has to be rewritten in the different frameworks is interesting since it could help UAS Europe AB, as well as other parties, plan and prepare for the work needed to make their respective applications available to a broader set of operating systems.
4. How does the different frameworks compare regarding how much and what type of platform specific code that has to be written to create a GCS application?
To make the applications work across different platforms some code might have to be written specifically for a certain platform. This could include, for example, code that distinguish between touch input for a smartphone and mouse input for a PC or code written only to adapt the user interface depending on the screen size.

The main aim for this thesis is to investigate UWP, Xamarin and Mono using above mentioned problem statement questions. Another aim is to create a prototype cross platform application that can be further extended into a full GCS software by UAS Europe AB in the future.

1.5 Method for this thesis

The general method used in this work is to evaluate the differences between UWP, Xamarin and Mono from a GCS software perspective. First, three small prototype GCS applications are created using the three cross-platform development frameworks. The three prototypes created are a UWP application targeting Windows 10 PCs and Windows 10 Mobiles, a Xamarin application targeting Windows 10 PCs, Windows 10 Mobiles, Android devices and iOS devices and a Mono application targeting Windows PCs and Linux PCs.

With the problem statement in mind, these prototypes are then compared using a table of functional requirements, a resource usage evaluation, an estimation on how much of the code and what parts from SkyView GCS that can be reused in the new application and by comparing how much code that had to be written specifically for each platform, in each application.

1.6 Delimitations

There are several different approaches on how to create mobile applications. They are explained by for example Latif et al. [1]. This thesis however will only focus on the cross-compiled approach using Mono, Xamarin and UWP.

Developing a fully working application takes a great deal of time. The time available during this thesis is not enough to develop a complete application, neither is it enough to rewrite all the parts of SkyView GCS to work across multiple platforms.

1.7 Thesis overview

This section aims to give the reader an idea about which different chapters this thesis consists of and what content to expect in the different chapters.

This first chapter introduced and motivated the thesis. It also presented the problem statement questions that will be answered and gave some examples on delimitations.

The next chapter, (Chapter 2), contains background information regarding UAS Europe AB and their existing GCS solution called SkyView GCS. It also presents the functional requirements that will be used to compare the different prototype applications.
In Chapter 3, the theory behind the thesis is presented. This includes theory about the Common Language Infrastructure and how the .NET Framework, UWP, Xamarin and Mono are related. It also includes relevant earlier work within the subject.

Chapter 4 explains how the work in the thesis was done and how it relates to the problem statement questions. This chapter can be seen as an extension to Section 1.5.

After that, Chapter 5 presents the results obtained by applying the method from the fourth chapter. The results are also briefly discussed.

In Chapter 6 the method and the results are discussed in more detail. Chapter 6 also contains a section about the work in a wider context in which possible positive and negative effects on society and environment are discussed.

In the last chapter, (Chapter 7), the work is concluded by providing answers to the problem statement questions. This chapter also give suggestions on further work that could be interesting to examine.
This chapter aims to provide the background for this thesis. In Section 2.1 the company UAS Europe AB is described and in Section 2.2 their software for monitoring and controlling unmanned aircrafts is presented. After that, in Section 2.3 UAS Europe AB’s most common UAV is introduced and finally in Section 2.4 UAS Europe AB’s initial functional requirements for the prototype applications of this thesis are presented.

2.1 UAS Europe AB

UAS Europe AB is a small company founded in Linköping in 2011. They are working with everything from developing and repairing UAVs to developing software and hardware related to monitoring and steering the UAVs. UAS Europe AB’s main products are their ground control station SkyView GCS, their UAVs Agri Owl and Spy Owl and their flight controller EasyPilot 3.0. They also have some other projects including hexacopters and education of customers.

2.2 SkyView GCS

SkyView GCS is UAS Europe AB’s ground control station solution. It consists of, among other things, a portable computer including the SkyView GCS software, long range antennas, a video server and various joysticks and buttons. For an image of the SkyView GCS hardware see the GCS hardware part of Figure 1.1. The rest of this section will focus on the SkyView GCS software.
2.2. SkyView GCS

The SkyView GCS software is a .NET Framework application, written using mainly C#. It contains about 20 different components. In this thesis these components have been divided into 5 different software categories that can be seen in Figure 2.1. These software categories are listed and explained below:

- **Core code**
  The core code includes several base classes for GUI components and settings as well as some generic classes used by the other components. This category contains approximately 35,000 lines of code.

- **Visualizer**
  The visualizer includes a map and a 3D view. It is responsible for showing data such as the position of one or more UAVs on the map as well as flight paths and planned missions. The visualizer component (both 3D view and map) can be seen in the middle of the screen in Figure 2.2. The visualizer contains about 31,000 lines of code.

- **Displays**
  The displays include various windows for displaying information such as speed, engine power and signal strength. These can be seen in the bottom of Figure 2.2. It also includes a flight director system, giving the operator information about the roll, pitch and heading of the UAV. The flight director can be seen in the upper right corner of Figure 2.2. Approximately 30,000 lines of code are included in this category.

- **Video component**
  The video component handles communication with the video server as well as displaying and recording the video. It can be seen in the top left part of Figure 2.2. The video component contains approximately 2000 lines of code.

- **Communication**
  The communication component contains code for communicating and translating data to and from the UAV. It is created to follow the NATO STANAG 4586 protocol for communication with UAVs. This allows it to be used with a wide range of UAVs from several different UAV manufacturers. The communication component is the largest part of SkyView GCS and contains around 43,000 lines of code.
Since the SkyView GCS software is developed using the .NET Framework it can, in its current state, only be used on Windows PCs. This is the main reason why UAS Europe AB is interested in the subject of this thesis, to understand what the possibilities and limitations are when creating a GCS software using the cross platform approaches UWP, Xamarin and Mono.

2.3 Agri/Spy Owl

UAS Europe AB’s two main UAVs are called Agri Owl 200 (see Figure 2.3) and Spy Owl 200. They have a wingspan of about 2 meters and can fly for approximately 3 hours at a cruise speed of about 20 meters per second. The real difference between them is that Agri Owl is more focused on agricultural applications, having a multispectral camera suited for images of agricultural land, while Spy Owl is generally equipped with a stabilized camera and an IR-camera.
2.4 Functional requirements

Prior to this project, UAS Europe AB had made a list of the most essential functionalities that a GCS software should provide. The functionalities are presented with some minor modifications in the following list. These are the functional requirements that were tested for the UWP, Xamarin and Mono prototype applications. They are ordered by priority.

1. Show a map with custom icons
2. Draw lines and polygons on the map
3. Show multiple panels with content
4. Support offline usage
5. Work with different screen sizes
6. Work with both touch and mouse input
7. Support writing and reading files to and from local memory
8. Send and receive data using WiFi
9. Import and show different layers on the map
10. Play a video from a file
11. Show a 3D view of the map and the icons
12. Stream video from UAS Europe AB’s video server using the Real Time Streaming Protocol (RTSP)
13. Save settings such as type of map and initial zoom level between usage session
14. Load GUI components dynamically from a .dll file
15. Show the icons in a 3D environment with the possibility to show other objects in 3D as well

Drawing lines on the map is important since it is likely that the user of a GCS will want to add a flight mission with multiple waypoints for the UAV to visit. Immediately drawing a mission using a line is a very intuitive way of creating such a mission. Another use case for the GCS operator would be to create a mission were the UAV is supposed to scan an area of land. In such case, being able to draw a polygon representing the mission area instead of a line is beneficial.

Sometimes it is necessary to use a GCS in a remote environment, maybe without access to the Internet. In such cases it is important that the user of the GCS has the possibility to download map data in advance and then use it without Internet access.

Some of UAS Europe AB’s customers have their own GUI components developed to suit their specific UAV. For these customers it is important that they can use their components in the GCS by loading them dynamically at runtime using a .dll file.
In this chapter the theory behind the thesis is explained to give the reader a better understanding of the method choices in Chapter 4 and the results in Chapter 5. The biggest focus will be on the .NET Framework, UWP, Xamarin and Mono and how they are related to this thesis.

To understand what the .NET Framework, UWP, Xamarin and Mono are and how they are related, it is first important to understand what the Common Language Infrastructure (CLI) is. Therefore the CLI is presented first in Section 3.1. After that, the .NET Framework, UWP, Xamarin and Mono are presented in Section 3.2. After that, a standard for developing cross platform applications called the .NET Standard is presented in Section 3.3. The chapter is ended with a short summary of related work and other relevant material to this thesis in Section 3.4.

Much of the theory in this thesis is related to Figure 3.1 and the text in this chapter will on a number of occasions reference back to that figure.
3.1 The Common Language Infrastructure

The Common Language Infrastructure (CLI) presented by Microsoft provides a standard for how code should be executed in different operating systems. This includes, for example, a specification called the Common Type System defining how data types should be stored in memory as well as metadata defining how the code should be structured. It also provides exception handling, garbage collection and functions for security and interoperability. In the first sentences of the ISO standard about the CLI, the following is written [4].

"ISO/IEC 23271:2012 defines the Common Language Infrastructure (CLI) in which applications written in multiple high-level languages can be executed in different system environments without the need to rewrite those applications to take into consideration the unique characteristics of those environments."

Examples of languages which conform to the CLI are C#, Visual Basic .NET and F#. These all have their individual compiler that turns written code into byte code called Common Intermediate Language. There are also several other languages which conform to the CLI but this thesis will only focus on the C# programming language. The CLI can be seen at the bottom of Figure 3.1 and acts as a common base for the CLI implementations.
3.2 Common Language Infrastructure implementations

On top of the CLI there are a number of CLI implementations. Those that are relevant to this thesis are presented in the following sections and can be seen in Figure 3.1.

3.2.1 .NET Framework

The .NET Framework is the oldest CLI implementation dating back to 2002. A big part of the .NET Framework is the framework class library. This Application Programing Interface (API) contains a number of classes and interfaces that give the developer access to system functionality. This is intended to make it easier for developers to create their applications since it provides access to services such as network communication, file input and output and multithreading [5]. These components are arranged in different namespaces depending on their application. For example the System.Net namespace for network communication, System.IO for writing and reading data to streams and System.Threading for multithread programming.

On top of the .NET Framework class library are the application models. The .NET Framework includes several different application models which contain components and Graphical User Interface (GUI) controls specific to that application type. For example, Windows Forms includes GUI components such as buttons, labels and scroll bars. Windows Presentation Foundation is similar to Windows Forms but is updated to use DirectX and ASP.NET is more focused on web applications and contains components for HTML and web hosting. Using the .NET Framework class library and application models, applications can be created for the Windows operating system.

One such application is SkyView GCS which is created using components from the Windows Forms application model and components from the .NET Framework class library.

3.2.2 .NET Core

.NET Core is another implementation of the CLI. It was released in 2016 so it is much more recent than the .NET Framework. The .NET Core class library is fairly similar to the .NET Framework class library but with its own implementation of components. The main difference to the .NET Framework is that .NET Core is open source and available for several other platforms including Windows, Linux and macOS. On the other hand, .NET Core does not have any of the components available in, for example, Windows Forms and Windows Presentation Foundation.

UWP
UWP is an application model using the .NET Core implementation of the CLI. UWP is used for developing applications for devices running some version of Windows 10. This includes Windows 10 desktops, mobile devices, Xbox gaming consoles and several other types of devices, which can be seen in Figure 3.2.
3.2. Common Language Infrastructure implementations

To be able to run the same application on such a wide range of devices, UWP automatically scale user interface components to suite screen size. The developer is also given tools to reposition and alter the way content is displayed on different device families.

UWP developers can use components available in the .NET Core class library to create their applications as well as an API specifically for UWP. These APIs provide, for example, network communication, threading and various user interface and map components. While UWP applications work on most of Microsoft’s newer devices there is no support for running UWP applications on earlier versions of Windows such as Windows 8 or Windows Phone.

As mentioned in Section 1.5, one of the prototype applications created in this thesis will be based on UWP. It can be seen in Figure 3.1.

3.2.3 Mono

Mono started as an open source project in 2004 and has even more similarities to the .NET Framework than .NET Core has when it comes to the class libraries. In fact, Mono is developed to be as similar to the .NET Framework as possible while still targeting other operating systems than just Windows, such as Ubuntu, Debian, CentOS and Raspbian. It has its own class library that provide almost the same components that the .NET Framework class library provide, excluding some Windows-specific components.

Mono also contains most of the Windows Forms namespace available when developing Windows Forms applications. However, none of the Windows Presentation Foundation components are available in Mono. Since .NET Framework and Mono are so similar, often applications can be easily converted from the .NET Framework to Mono. It is common that the program does not even have to be recompiled and can be run directly using Mono.

As mentioned in Section 1.5, one of the prototype applications created in this thesis will be based on Mono. That application will be a Mono, Windows Forms application as seen in Figure 3.1.

Figure 3.2: Devices that can run applications written for UWP [6].
Xamarin

Xamarin is an application model in the Mono implementation of the CLI. It is more focused on Mobile development than the other application models presented so far with native implementations for iOS using Xamarin.iOS and Android using Xamarin.Android. It also has support for creating UWP and Mac applications. This allows developers to write applications using a single set of available components for both iOS, Android, Mac OS and Windows 10.

Xamarin applications are typically divided into multiple different so called projects. One project with shared code that can be used on all the targeted platforms. This project contains the business logic of the application and is represented at the bottom of Figure 3.3. They also contain separate projects, one for each of the applications target platforms. So, for example, one project for iOS, one project for Android and one project for UWP. These projects typically contain platform specific code such as GUI components, layouts and code that access native functionalities. This is represented as the purple, green and blue boxes at the top of Figure 3.3.

Figure 3.3: Overview of how code is structured in Xamarin.

In 2014 Xamarin.Forms was introduced as a shared set of GUI components for iOS, Android and even UWP. This allow GUI code to be reused across the platforms as well (seen as the box in the middle of Figure 3.3). For example, in Xamarin.Forms there is a GUI component for displaying text which is connected to the native GUI component in iOS, Android and UWP. Instead of using the native component separately for each platform, the Xamarin.Forms component can be used in the shared code. Since Xamarin.Forms is open source all of the GUI components and their connections to the native components of each platform can be found online. The connections can also be changed by a developer when needed.

As mentioned in Section 1.5, one of the prototype applications created in this thesis will be based on Xamarin. That application will be created for Windows 10 PC and Windows 10 Mobile (by targeting UWP), Android and iOS as seen in Figure 3.3. It will also use several components from the Xamarin.Forms library of GUI components.
3.3 .NET Standard

Since several of the class libraries of the different CLI implementations are similar it becomes possible to write code that works across several different CLI implementations. To make sure that all the class libraries used when developing for multiple CLI implementations are available, some way of keeping track of available libraries is required. This has traditionally been done using Portable Class Libraries (PCL). In a PCL a subset of all libraries are available depending on what platforms are chosen as a target of the PCL. For example, if .NET Framework 4.6, Xamarin.Android and Xamarin.iOS are chosen as targets, only the libraries available to all of those three becomes available. The large amount of possible combinations of target platforms lead to several different versions (profiles) of PCLs, making it difficult to know whether a PCL would be compatible with a set of platforms or not. Also, it became difficult to know if more platforms could be targeted without breaking the existing code.

In 2016 Microsoft announced a new way of sharing code across CLI implementations called the .NET Standard which can be seen as another layer between the CLI and the CLI implementations (see Figure 3.4). The purpose of the .NET Standard is to reduce the difficulties that the PCLs had. Instead of making a set of APIs available depending on an intersection of platforms the .NET Standard contains a number of different versions. At the time of writing the versions 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6 and 2.0 are available, all defining their own set of API classes.

![Diagram of .NET Standard](image)

Figure 3.4: The .NET Standard’s relation to the CLI and the CLI implementations.

The different .NET Standard versions are then implemented by the different CLI implementations. For example, the .NET Framework version 4.6 implements the .NET Standard version 1.3 just as Xamarin Android 7.0 and UWP 10.0 does. So instead of targeting multiple different platforms with a PCL, a version of the .NET Standard is targeted and all CLI implementations that implement that standard are guaranteed to be able to use that code. This makes it easy to create cross platform code and keep track of which platforms are supported.
3.4 Related work

In this section some of the earlier work and research related to this thesis are presented. The first subsection contains difficulties and advantages with multi-platform development. After that, research related to Xamarin, UWP and Mono is presented, including earlier attempts on moving from the .NET Framework to Mono as well as research related to creating and evaluating Xamarin. Lastly, a paper about a GCS software for Android is described.

3.4.1 Multi-platform application development

The subject of how to write code once and run it on several different types of devices has been studied for almost 40 years. However, it has seen a big increase in popularity in the last 10 years, especially with the increasing popularity of smartphones.

Joorabchi et al. [10] investigated the biggest challenges in mobile application development by interviewing senior mobile developers. The investigation also includes a survey with respondents from the mobile developer community. Two of the biggest challenges, according to the participants, were the lack of cross platform automated testing tools and the difficulties of writing user interface for multiple platforms without compromising either consistency or platform design standards.

Mercado et al. [11], have studied the relationship between mobile application development approach used and the number of user complaints received. They have compared native applications with applications created using an interpreted/generated approach (were Xamarin is included) and applications created using a hybrid approach. The comparison was done by analyzing over 700,000 user reviews for 50 different mobile applications and categorizing the complaints into one of four different quality concerns: performance, usability, security and reliability. They found that hybrid applications in general receive more complaints than native and interpreted/generated applications and that the most common complaint from the users were related to reliability. Something that should be noted however is that the study only considered 50 different applications, and only three of them were native applications. This could be considered to few to really compare the different approaches.

Boushehrinejadmoradi et al. [12], Raj and Tolety [13], Charkaoui et al. [14] and Latif et al. [1] write about the web-based, native, hybrid and cross compiled approach to mobile application development. They all agree on the advantages of being able to reuse code across the different platforms. Latif et al. [1] also write about the importance of being able to access native features on the different platforms. Relevant to this thesis, one such native feature is the devices built in map. For Android devices this is a Google Maps application while iOS devices use Apple Maps. Being able to use these map components are really relevant to the GCS prototypes in this thesis since they all will use maps for displaying the location of UAVs.

3.4.2 Xamarin, UWP and Mono

In a bachelor thesis by Berglind and Larsson [15], the development process using Xamarin is investigated. According to Berglind and Larsson, some of the GUI in their application had to be rewritten for each platform while most of the business logic could be reused. This information helps when planning the development of the prototype in Xamarin.
3.4. Related work

Delia et al. [16] performed a comparison between different cross platform development methods including Xamarin and came to the conclusion that using Xamarin, a fully native Android and Windows 8 application could be created. The code reuse for their application was about 50%. The reason a higher code reuse could not be achieved was because of the need for writing separate GUI code for the applications. However, it is important to note that they did not use Xamarin.Forms in their study which probably affected the amount of code reuse.

Redda [17] writes about cross-platform development using a number of different cross platform frameworks including Xamarin. One of his several conclusions is that Xamarin produces well-organized and flexible code due to the separation of the code for different platforms into different files. This is something that Berglind and Larsson also pointed out as a great feature of Xamarin.

Nishimura and Timossi [18] write about the development of a control system using .NET Framework and also about running it on Linux using Mono. Their experience is that GUI parts, such as Windows Forms components, of the .NET Framework, are not as well supported as most non GUI components. They also write that they had problems with some 3:rd party libraries not being supported in Mono. The conclusion is that Mono is, in most cases, a valid way of making .NET Framework applications available to different platforms, such as Linux.

Knittl [19] evaluates the differences between Mono and .NET Framework in his master thesis. According to Knittl, Mono miss some of the Windows specific features that .NET Framework has. This is not a surprise since Mono is a cross-platform framework. He also discusses the differences between the Windows Forms implementation in Mono and in the .NET Framework. His conclusion is that "there are still rough edges in the implementation provided by Mono". He also mentions differences regarding for example security. Knittl also compared the resource usage for applications created using Mono and the .NET Framework. He found that in all his tests the .NET Framework application outperformed the Mono application. In six out of seven cases, the .NET Framework application had less than half the execution time of the Mono application. However, it is important to note that this evaluation was made in 2012 using Mono 2.11 and some things might have changed since then, both regarding available libraries and performance.

The books by Hermes [20], Moemeka and Moemeka [21] and Easton and King [22] are about application development in general using Xamarin, UWP and Mono respectively. Also, the book Microsoft Mapping by Au and Rischpater [23], which is specifically about mapping using Bing Maps, contains several details on how to create applications featuring a map in UWP. These books are really relevant to the development of the three prototypes.

3.4.3 Mobile GCS software

Do, Kwon and Moon [24] write about an implementation of a ground control software aimed at the Android platform using Java and make an evaluation of the same software. Their study show that this kind of software has high potential in UAV-related systems. One of the reasons for this is that the light weight hardware of mobile devices can be beneficial when using such a software. Also, the rich user interface of the Android operating system was mentioned as advantageous to a GCS software.
In this chapter the method used for this thesis is presented to give the reader a better understanding about how the results in this thesis have been achieved. As mentioned in Section 1.3, three different prototype applications was created in this thesis using UWP, Xamarin and Mono respectively. There is a simple reason behind that approach. A prototype application is needed for each cross platform approach in order to conduct the tests required to answer the questions raised under Section 1.4. The prototype applications was then evaluated based on the functional requirements, which was described in Section 2.4, code reuse estimations and resource usage. These prototypes try to replace the part of the SkyView GCS software represented by the green box in Figure 2.1 which is used as the baseline. All applications is developed using Visual Studio 2017 Community 15.4.0 Preview on a Windows 10 computer with an Intel Core i5-3427U processor and 8 GB RAM memory. For the rest of this thesis that computer will be referred to as the "Windows 10 PC".

In Sections 4.1-4.3 the method conducted and the tools used when developing the UWP, Xamarin and Mono application are presented. The purpose of these subsections is to give an overview of how the applications were created and which platforms the applications were tested on. Another focus in these sections is to explain what parts of the UWP, Xamarin and Mono prototypes correspond to the parts of the SkyView GCS software in the green box in Figure 2.1.

The method for comparing the applications, using the functional requirements presented in Section 2.4 is described in Section 4.4. Some of the functional requirements was tested for all three prototype applications while some of the functional requirements was only tested using Xamarin. The main reason behind comparing how well UWP, Xamarin and Mono can fulfill the functional requirements from Section 2.4 is to be able to answer the first problem statement question "To what extent does UWP, Xamarin and Mono support the functionality needed for creating a Ground Control System software aimed at mobile devices?".
Next, in Section 4.5 the approach when comparing the UWP, Xamarin and Mono prototype applications with regard to their code reuse across their target platforms is explained. The reason behind the work described in this section is to try and answer the problem statement question "How does the different frameworks compare regarding how much and what type of platform specific code that has to be written to create a GCS application?".

In Section 4.6 the method used when finding out how much code from the original SkyView GCS software can be reused in UWP, Xamarin and Mono respectively. This section is closely related to the problem statement question "Having a C# .NET Framework application, are there any typical segments of code that has to be rewritten in order to work on the different operating systems, using the different frameworks?".

Finally in Section 4.7 the method related to the resource evaluation of the prototype applications is presented. The reason behind this section is to explain how the problem statement question "In what ways does a Ground Control Software developed using UWP, Xamarin and Mono differ regarding CPU and memory usage?" was answered.

4.1 UWP application

This section describes the method used when creating the UWP application. The presentation of the platforms that the application was tested on is followed by a description of the different parts of the UWP application and their connection to the parts in the green box in Figure 2.1.

4.1.1 Platforms and tools

As mentioned in Section 3.2.2 there are a wide range of devices that can run applications created using UWP. However, devices such as Xbox and IoT-devices are generally not suitable for GCS applications. Testing the prototype application on such devices would certainly be interesting but not really relevant for a GCS application. Instead, the following two platforms have been chosen for testing the UWP application.

- **Windows 10 PC**
  The application was tested using the Windows 10 PC mentioned in the beginning of this chapter. The screen used to display the application had a resolution of 1920x1080 pixels.

- **Windows 10 Mobile**
  The application was tested using a Windows 10 Mobile emulator running on the Windows 10 PC. This is a software created by Microsoft to emulate a real Windows 10 Mobile device. The emulator used when testing the application emulates a 5 inch 720p screen and 1 GB RAM.

When testing the functional requirement "The application should work with different screen size’s" another Windows 10 Mobile emulator was used as well. That emulator had a 1080p 6 inch screen instead of the 720p 5 inch screen of the emulator described above.
4.1.2 Components

UWP has several built-in components that can be used to create a GCS software. Below, the components that were chosen for the Video component, Displays, and the map and 3D view in the UWP prototype software is presented.

- **Video component**
  To replace the video component in the SkyView GCS software a UWP component called MediaElement was used. It is a built-in component that can play and show video of various formats on the screen.

- **Displays**
  To replace the displays of the SkyView GCS software a couple of different UWP components were used. To replace components that show information about the UAV using just plain text, a UWP component called TextBlock was used. The flight director (shown in the top right in Figure 2.2) was replaced using UWP image components that were rotated and moved up and down according to the pitch and roll of the UAV.

- **Visualizer (map and 3D view)**
  A built-in UWP component called MapControl was used to replace the SkyView GCS visualizer. It uses maps and map data from Bing maps. UWP components such as MapIcons and MapPolylines can then be added to the map to display for example UAVs and flight paths.

4.2 Xamarin application

This section contains a description of the method used when creating the Xamarin application. First the platforms that the application was tested on are presented. After that, the different parts of the Xamarin application and their connection to the parts in the green box in Figure 2.1 are presented.

4.2.1 Platforms and tools

Xamarin supports several different platforms which were shown in Section 3.2.3. This thesis focus on the following platforms:

- **Windows 10 PC**
  The application was tested using the Windows 10 PC mentioned in the beginning of this chapter.

- **Windows 10 Mobile**
  The application was tested using a Windows 10 Mobile emulator running on the Windows 10 PC. This is a software created by Microsoft to emulate a real Windows 10 Mobile device. The emulator used when testing the application emulates a 5 inch 720p screen and 1 GB RAM.

- **iOS**
  The application was tested using an iPhone 5S running iOS 11. This device has a 4 inch screen and 1 GB RAM.

- **Android**
  The application was tested using an Motorola Nexus 6 running Android 7.1 Nougat. This device has a 6 inch screen and 3 GB RAM.
When testing the functional requirement “The application should work with different screen sizes” another Windows 10 Mobile emulator was used as well. That emulator had a 1080p 6 inch screen instead of the 720p 5 inch screen of the emulator described above.

Because of limitations from Apple it is not legally possible to directly test applications developed using a Windows computer on, for example, an iPhone. Instead, this had to be done either through a Mac or by using an iOS app called Xamarin Live Player. In the test environment no Mac was available, so instead the Xamarin Live Player app was used to test the Xamarin application on the iPhone 5S. A number of limitations with this approach were discovered and these are discussed in section 6.1.

4.2.2 Components

The shared code for the Xamarin application was developed in a single development project, using components available in the .NET Standard 1.4 and GUI components from the Xamarin.Forms library. This ensures that the code can be reused on both Windows 10 devices, devices running Android 7.0 or later and devices running iOS 10.0 or later [25].

The platform-specific code was divided into 3 different subprojects as seen in Figure 4.1. One UWP project for Windows 10 PC and Windows 10 Mobile, one Xamarin Android project for Android and one Xamarin iOS project for iOS. These separate subprojects contain platform specific code that is only used by the respective platform.

As mentioned earlier in this section and seen in Figure 4.1, the Xamarin application contains a UWP project. This means that some of the code from the UWP application and the Xamarin application overlaps. However, it is still interesting to run the application on a Windows 10 PC and a Windows 10 Mobile in the Xamarin application to see how they differs compared to the pure UWP application.

The Xamarin application is using several GUI components from the Xamarin.Forms library. The following list shows what components were used for the Video component, the Displays, and the map and 3D view in the Xamarin prototype.
• **Video component**
  There is no video component in Xamarin.Forms. Instead, a new video component was created and implemented for the UWP and the Android part of the Xamarin application. In the UWP implementation the same MediaElement component as in the UWP application was used, and the Android implementation used an Android component called VideoView.

• **Displays**
  Xamarin has several components that could be used for replacing the displays of the SkyView GCS software. To replace plain text fields, a Xamarin.Forms component called Label was used. To replace the flight director, a similar approach as in the UWP application was taken. Using Xamarin.Forms image components a new flight director could be created by rotating and moving an image around to show the pitch and the roll of a UAV.

• **Visualizer (map and 3D view)**
  To replace the SkyView GCS visualizer, the built-in Xamarin.Forms map component was used. This map component differs depending on which platform the application is running on. On Windows devices it uses the same MapControl component that the UWP application uses. On Android devices it uses a component called GoogleMap that exists in Android and uses maps and map data from Google Maps. On iOS it uses a component from iOS called MKMapView which uses Apple Maps for maps and map data.

### 4.3 Mono application

In this section the method used when creating the Mono application is explained. First the platforms that the application was tested on are presented. After that, the different parts of the Mono application and their connection to the parts in the green box in Figure 2.1 are presented.

#### 4.3.1 Platforms and tools

As mentioned in Section 3.2.3 Mono applications can run both on Windows and various Linux distributions. This thesis focus on the following platforms:

• **Windows 10 PC**
  The application was tested using the Windows 10 PC mentioned in the beginning of this chapter.

• **Ubuntu**
  The application was tested using Ubuntu 16.04.3 running inside a virtual machine using a tool called VirtualBox on the Windows 10 PC. The Ubuntu installation was given 1GB RAM and used the same 1920x1080 screen that the Windows 10 PC used.

While there are several Linux distributions other than Ubuntu that Mono supports, it was decided that only Ubuntu should be tested. The main reason behind this was that Ubuntu is arguably the most common Linux distribution. In a survey by Stack Overflow [26], Ubuntu was the most popular Linux distribution for desktop computers followed by Fedora, Mint and Debian. Also, Ubuntu, Fedora, Mint and Debian are very similar so Ubuntu should be a good representative for all of them.

The Mono version chosen in this thesis is Mono 5.2.0 which was the most recent Mono version when the work on this thesis started.
4.4. Testing functional requirements

4.3.2 Components

The Mono application was created using mainly components from Windows Forms. The Video component, the Displays and the map and 3D view was created in Mono by using the following components:

- **Video component**
  One way of creating a video component in a Windows Forms application is to use a Windows Media Player component. However, Windows Media Player cannot be used in Linux and since the Mono application had to work on Linux as well, this component could not be used. Instead, a new component using a cross-platform media player called MPlayer was created and used.

- **Displays**
  The Windows Forms library contains several components for displaying data. Normal text data, such as altitude and speed, was displayed in the Mono application using Label components and just like in the Xamarin and UWP application, image components were used to create a flight controller displaying pitch and roll.

- **Visualizer (map and 3D view)**
  Mono does not have any built-in map component like UWP and Xamarin has. Instead, an open source map component, that can be used in Mono Windows Forms applications, called GreatMaps was used. This component has support for multiple different map providers including Bing Maps, Google Maps and Open Street Map. In this thesis Bing Maps was chosen as the map provider for the Mono application. In GreatMaps icons, lines and polygons can be added to the map using the components GMapMarker, GMapRoute and GMapPolygon.

4.4 Testing functional requirements

This section contains a description of the method for testing the functional requirements from Section 2.4. In general, the following three questions were used when determining how well the different platforms support the functionalities.

1. **Is there immediate support for this functionality?**
2. **Is there any third party library available that provides this functionality?**
3. **Can a new implementation to this functionality be estimated to be done in less than 50 hours?**

If the answer to the first question is yes and the testing of the functionality, using the prototype, on the different platforms shows that the functionality is indeed realised, the functionality is marked with ✓.

If the answer to the second or the third question is yes and the testing of the functionality, using the prototype, on the different platforms shows that the functionality is indeed realised, the functionality is marked with ✗.

If the answer to all three questions is no, the functionality is marked with ✗.

These results were then saved and summarized in tables showing all the functional requirements from Section 2.4 and all the framework and platform combinations studied. Two tables were used to display the results. One table comparing UWP, Xamarin and Mono for functional requirement number 1 through 11 and one table evaluating Xamarin for functional
requirement number 12 through 15.

50 hours was chosen as a limitation since there were 800 hours available in this thesis and 15 functional requirements. 800 divided by 15 is approximately 50. Spending 50 hours on every functionality however would not have been possible since then there would have been no time to, for example, write this report or evaluate code reuse. However, an assumption that most functionalities would take considerably less time was made.

4.5 Evaluating code reuse across platforms

To estimate how much code an application developer has to write specifically for each different platform in UWP, Xamarin and Mono, a comparison of the amount of reusable code was made. This was done by counting the number of lines of code for just a specific platform in UWP, Xamarin and Mono respectively and then comparing that to the number of lines of code that were shared between the platforms within the framework.

For example, in UWP this meant that the number of lines of code that was written only for Windows 10 PC and the number of lines of code that was written only for Windows 10 Mobile was counted. These numbers were then saved in a table. The same was also done for Xamarin and Mono.

Not all files can be included when counting the number of platform specific lines of code. To decide which files to include in the comparison, the files were divided into three different categories:

- **Manually written files**
  This category includes files were the code was written manually during the work on fulfilling the functional requirements. These files are counted in the comparison.

- **Files with automatically generated code that is the result of designing a GUI**
  Developing a GUI can be done both by writing code directly and by dragging and dropping components in a designer and have the designer automatically create the code. Such GUI files are still the result of the work done by an application developer and are therefore included in the comparison.

- **Files that are automatically generated without any noticeable work done by a developer**
  This category includes files that are only used by the compiler but are still visible and explorable by the developer in Visual Studio. One example is a file called Resource.Designer.cs that exists in Xamarin Android projects. This file contains information about all the resources available to the application, such as images, fonts and colors. Even with an empty project this file contains thousands of lines of code. This category also includes files that are generated by the compiler when compiling the application. Files from this category are not counted in the comparison.

The number of lines of code were then saved in a table and the percent of code that was shared for each framework and platform was calculated.
4.6 Evaluating code reuse from SkyView GCS

To assess the amount of the code in the SkyView GCS software that could potentially be reused in UWP, Xamarin and Mono, an evaluation using a tool called the .NET Portability analyzer was performed. The .NET Portability analyzer is a console application that analyzes what libraries from the different class libraries are used in the code. It then produces a report with an estimation about how much, as a percentage, of the code that can be immediately reused. In some cases it also gives suggestions on how to change the code so that it can be used in the different platforms analyzed. The .NET Portability analyzer also exists as a Visual Studio plugin which is what was used in this thesis.

The .NET Portability analyzer has support for analyzing portability to several different CLI implementations such as different versions of .NET Core, .NET Framework, Mono and .NET Standard.

The SkyView GCS software consists of over 20 different sub-projects. To make it easier to understand what parts of the Skyview GCS software are expected to be easy to reuse, these projects were divided into 6 subcategories.

- **Core code**
  The core code category mainly contains projects with base classes and generic classes used by the other projects. The core code category contains about 35000 lines of code.

- **Visualizer**
  The visualizer category includes projects related to the map and the 3D view of SkyView GCS. The visualizer category contains about 31000 lines of code.

- **Displays**
  This category contains projects related to the displays. This includes various windows for displaying information such as speed, engine power and signal strength. It also includes the flight director, showing the pitch and roll of the UAV. The displays category contains about 30000 lines of code.

- **Video**
  The video category contains projects that handle communication with the video server as well as displaying and recording the video from the UAV. The video category contains about 2000 lines of code.

- **Communication**
  The communication category contains projects that are connected to communicating and converting data to and from the UAV. The communication category contains about 43000 lines of code.

- **Other**
  In this category the projects that did not fit in any of the other categories were placed. The Other category contains about 4600 lines of code.

These are the same categories used to describe the SkyView GCS software in Section 2.2 with the only exception being the "Other" category. The categorization was done together with a SkyView GCS expert at UAS Europe AB. The different projects were then analyzed using the .NET Portability analyzer and the percentages of reusable code for the categories was calculated and saved in a table in Excel.
4.7 Evaluating resource usage

The CPU and memory usage of the prototypes was evaluated using the built-in performance profiler in Visual Studio for the UWP application and the UWP part of the Xamarin application, the Windows Performance Monitor for the Windows Mono application, the System Activity Report (SAR) for the Linux Mono application, and Android Monitor for the Xamarin Android application.

Before starting the resource evaluation, all functionality that would not be used during the resource evaluation was removed from the UWP, Xamarin and Mono prototype. This included removing video components, panels and displays, flight director and other components. The reason behind this was to make the test as fair as possible by running only the common denominator.

The resource evaluation was designed to reflect typical usage of a GCS software and was automated using built-in timers for the UWP and the Mono application and a custom created timer for Xamarin. The steps involved in the resource evaluation are described below and shown in Figure 4.2.

- **Start up**
  The application takes a couple of seconds to start up. During this time the map was initialized and positioned over the first location and the UAVs were created.

- **Move UAVs**
  Approximately 10 seconds into the evaluation two UAVs started to move. For the next 10 seconds they were moved 10 times every second in different directions.

- **Draw lines**
  After approximately 30 seconds, polylines started to be drawn on the map. During the following 10 seconds, 10 lines, each containing 10 vertices was drawn on the map.

- **Change location**
  About 10 seconds after the lines had been drawn, the location of the view of the map changed four times. The change happened once every 10 seconds. The locations that was used in the evaluation are described later in in this section and shown in Figure 4.3.

- **End of evaluation**
  After approximately 90 seconds the test was over and the program was closed.

The following figure shows in detail when each evaluation event occurred to give a better understanding of the evaluation and the results.

![Figure 4.2: Time-line showing when each evaluation-event is planned to occur.](image)

The locations to be shown in the resource evaluation were chosen specifically to be located quite far apart and with large differences in zoom levels. The reason for this was to evaluate if there were any differences between the applications regarding the location and zoom level of the map.
4.7. Evaluating resource usage

The following locations were chosen for the evaluation:

2. Louisville, USA. Latitude: 38.27°, Longitude: -85.79°, Altitude: ~100 meters.

![Image: Locations used for the resource evaluation]

Figure 4.3: The locations used for the resource evaluation. Linköping, Louisville, Siapa, Bass strait and Karthaus from top left to bottom right. All the images are from the UWP application using Bing Maps.

During the evaluation, data was collected using the performance measurement tools mentioned in the beginning of this section. The resource evaluation was made three times for each framework and platform to reduce the risk of invalid data. After that the data was evaluated and compared in Excel by creating graphs showing CPU and memory usage over time and by comparing the average CPU usage and the maximum memory usage for the different platforms.
In this chapter the results from the thesis are presented. First, in Section 5.1 the prototype applications created using UWP, Xamarin and Mono are presented.

In Section 5.2 the results from the comparison of how well the UWP, Xamarin and Mono applications fulfill the functional requirements introduced in Section 2.4 are presented. This is done using two different tables. One for the first set of functional requirements evaluated for UWP, Xamarin and Mono, and one for the second set of functional requirements evaluated only for Xamarin.

Next, the comparison of the code reuse is presented. This is done in two parts. First, in Section 5.3 the code reuse between the different operating systems, for the UWP, Xamarin and Mono application is presented. After that, in Section 5.4 the estimated expected code reuse from the SkyView GCS software using UWP, Xamarin and Mono is presented.

Finally, the chapter is ended with the results from the resource usage evaluation being presented in Section 5.5.
5.1 Prototype implementations

The three prototype applications created using UWP, Xamarin and Mono are all part of the result in this thesis. All three prototypes have the same general layout and can be seen in Figure 5.1.

![Figure 5.1: The prototypes created using Mono (top left), UWP (top right) and Xamarin (bottom). The applications are running on Linux, the Windows 10 PC and the Android mobile respectively.]

In the background is a map showing the different UAVs and some information about their positions. On the left part of the screen is a list with information regarding the mission status of all available UAVs. In the bottom center part of the screen are a couple of displays showing information such as altitude, speed and other UAV specific information as well as a flight director displaying roll and pitch. Lastly, in the top right corner is a video component the purpose of which is to display live video from the UAV.

5.2 Functional requirements

In this section the functional requirements that each framework and platform fulfill are presented. This is done using a table showing all the framework and platform combinations. In the table the functionalities have been marked with one of the three symbols ✅ (immediate support), ⌂ (third party support) or ❌ (no support). For a more detailed description of the symbols, see Section 4.4. First the functional requirements that were tested on UWP, Xamarin and Mono are presented and explained and after that the functional requirements that were tested only on Xamarin are presented and explained.

The focus of this chapter will mostly be on the functionalities that were not immediately or not at all supported by the respective frameworks and/or platform.
Table 5.1: Comparison of the different framework and platform combinations covering functional requirements. Acronyms used: W=Windows 10, WM=Windows 10 Mobile, A=Android, L=Linux

<table>
<thead>
<tr>
<th>#</th>
<th>Functional requirement</th>
<th>Framework and platform combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>W</td>
</tr>
<tr>
<td>1</td>
<td>Show a map with custom icons</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Draw lines and polygons on the map</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Show multiple panels with content</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>Use the application in offline mode</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>Work with different screen sizes</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>Work with both touch and mouse input</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>Save and open files of arbitrary format</td>
<td>✓</td>
</tr>
<tr>
<td>8</td>
<td>Send and receive data using WiFi</td>
<td>✓</td>
</tr>
<tr>
<td>9</td>
<td>Import and show different map layers on the map</td>
<td>✓</td>
</tr>
<tr>
<td>10</td>
<td>Play a video from a file</td>
<td>✓</td>
</tr>
<tr>
<td>11</td>
<td>Show a 3D view of the map and the icons</td>
<td>✓</td>
</tr>
</tbody>
</table>

Using the following list, each functionality will be explained. An empty space in the table means that the functionality could not be tested because of limitations in the testing environment for this thesis.

1. **Show a map with custom icons**
   This functionality is supported by all of the platforms. The reason why the Xamarin application got marked with ✗ was because the built in Xamarin map component did not have support for changing the icons. To do this some code has to be written separately for each platform.

2. **Draw polygons on the map**
   This functionality has the same support as the first functionality. It was possible to do on all platforms but in Xamarin some code has to be written for each platform.

3. **Show multiple panels on the map**
   All applications have support for this functionality across all platforms.

4. **Use the application in offline mode**
   The UWP application has immediate support for displaying offline maps of road type and the possibility to display other types of maps after some programming effort. The Xamarin application does not support road type maps like UWP do, but by writing some platform specific code the same level of functionality could be achieved. The support for offline maps in the Mono application is excellent. GreatMaps has immediate support for downloading and saving the map to a local database directly from the application.

5. **Work with different screen sizes**
   Both Xamarin and UWP have immediate support for scaling screen sizes using relative layout elements and scaling font sizes. The Windows Forms components in the Mono
application can be scaled by updating them as the screen size changed, but much more work than in the UWP and the Xamarin application was required.

6. **Work with both touch and mouse input**
   All application have support for this functionality across all platforms.

7. **Save and open files of arbitrary format**
   UWP and Mono have immediate support for saving and opening files of arbitrary format. In Xamarin a third party library called PCL Storage [27] was used.

8. **Send and receive data using WiFi**
   All platforms have support for sending and receiving data over WiFi. The sending and receiving of data was implemented using components from the System.Net.Sockets namespace, which exists in UWP, Xamarin and Mono. In the implementation done in this thesis, all of the applications sent the data using a User Datagram Protocol.

9. **Import and show different map layers on the map**
   The UWP application has support for importing and showing map layers on top of the map both from a local source and from an online source. In Xamarin the same was possible but it required a separate implementation for the UWP part and the Android part of the application. Mono and GreatMaps have no immediate support for displaying layers on top of the map.

10. **Play a video from a file**
    In UWP, video can easily be played from a local file using a component called MediaElement. Xamarin on the other hand, has no video component. Instead a new component was created which makes use of the same MediaElement component for the UWP part of the Xamarin application and an Android component called VideoView for the Android part of the Xamarin application. In Mono a component based on the cross platform media player called MPlayer was created.

11. **Show a 3D view of the map and the icons**
    In UWP the map can be tilted and the map icons can be given an altitude which makes them appear elevated from the ground. This was not the case in Xamarin Android. The reason Xamarin Android does not have support for elevated map icons is probably that the position of markers in Google maps is defined by a data type called LatLng which does not contain an altitude property [28]. Mono with GreatMaps is similar to Google maps in that it has no support for elevated markers.

As mentioned in Section 4.4, some tests were only done for the Xamarin prototype. The results from those tests are presented in the following table.

Table 5.2: The differences in functionality between the platforms in the Xamarin application. Acronyms used: W=Windows 10, WM=Windows 10 Mobile, A=Android.

<table>
<thead>
<tr>
<th>#</th>
<th>Functional requirement</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Stream video from UAS Europe AB’s video server using RTSP</td>
<td>☑ ☑ ☑</td>
</tr>
<tr>
<td>13</td>
<td>Save settings such as type of map and initial zoom level between usage session</td>
<td>☑ ☑ ☑</td>
</tr>
<tr>
<td>14</td>
<td>Load GUI components dynamically from a .dll file</td>
<td>☑ ☑ ☑</td>
</tr>
<tr>
<td>15</td>
<td>Show the UAVs in a 3D environment with the possibility to show other objects in 3D as well</td>
<td>☑ ☑ ☑</td>
</tr>
</tbody>
</table>
5.3 Code reuse across platforms

Using the following list, each functionality will be explained.

12. **Stream video from UAS Europe AB’s video server using the Real Time Streaming Protocol (RTSP)**
As seen in Table 5.1 playing video from a local file is possible in all Xamarin platforms. When changing the source of the video from a local file to a live RTSP video stream some changes had to be made for the UWP part of the Xamarin application. To make the UWP MediaElement compatible with RTSP streams a library called FFmpegInterop had to be used. The Android VideoView on the other hand can handle RTSP streams without changes to the application. However, it has approximately a three second delay when playing the video. That is the reason why all platforms were marked with ✓ in Table 5.2.

13. **Save settings such as type of map and initial zoom level between usage sessions**
Xamarin does not have immediate support for saving settings between sessions. Instead a plugin called Xam.Plugins.Settings was used. It has support for a wide range of platforms including UWP and Android.

14. **Load external GUI components dynamically from a .dll file**
Loading GUI components dynamically using a .dll file can be used to allow users of an application to load their own GUI components into the application at runtime. This is used in the SkyView GCS software. In UWP there is no support for loading external .dll files dynamically which is why both Windows 10 PC and Windows 10 Mobile has been marked with ✓ in Table 5.2. In the Android part of the Xamarin application this was not a problem and GUI components could be loaded into the application. However, since some platform specific code was needed the functionality is marked with ✓ in Table 5.2.

15. **Show the UAVs in a 3D environment with the possibility to show other objects in 3D as well**
As seen in Table 5.1 at requirement 11, the Xamarin Android application can not show a 3D view of the icons. The map in the UWP part of the Xamarin application on the other hand can even be configured to show 3D objects such as houses and bridges in large cities. This feature only works on the Windows 10 PC and not on the Windows 10 Mobile emulator.

### 5.3 Code reuse across platforms

In the following table the code reuse across the different platforms for UWP, Xamarin and Mono can be seen. In this section the table is explained and some of the reasons behind the results are shown.

<table>
<thead>
<tr>
<th>Framework and platform combinations</th>
<th>UWP</th>
<th>Xamarin</th>
<th>Mono</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform specific lines of code</td>
<td>0</td>
<td>0</td>
<td>699</td>
</tr>
<tr>
<td>Common lines of code</td>
<td>1208</td>
<td>776</td>
<td>1111</td>
</tr>
<tr>
<td>Common code percentage</td>
<td>100%</td>
<td>100%</td>
<td>52%</td>
</tr>
</tbody>
</table>

Table 5.3: The estimated amount of code that can be reused across the platforms. Acronyms used: W=Windows 10, WM=Windows 10 Mobile, A=Android, L=Linux
As can be seen in Table 5.3, the code that could be reused for the different platforms varied quite a lot between UWP, Xamarin and Mono. In UWP, no code at all was written to make the application work on both the Windows 10 PC and the Windows 10 Mobile. In Xamarin on the other hand, quite much code had to be written to make both the UWP and the Android part of the application functional.

Most of the code that had to be written separately in the Xamarin application was related to the map. The original map component in Xamarin.Forms does not have the functionality needed to create a GCS software so a great deal of code had to be written separately.

In the Mono application almost all the code could be used both on Linux and on the Windows 10 PC. The only exception was a small part of the video component implementation using MPlayer.

From the results of the evaluation of code reuse across platforms it is easy to see that a GCS software in Xamarin is likely to require a significant amount of platform specific code. On the other hand, the UWP part (699 lines) and the common part (776 lines) of the Xamarin application combined (1477 lines) contains only marginally more code than the UWP application (1208 lines). After that an additional 564 lines had to be written to make the application work on Android as well.

5.4 Code reuse from SkyView GCS

This section contains a presentation of the code reuse estimation from Skyview GCS. As explained in Section 4.6, the results were collected using a tool called the .NET Portability Analyzer.

Table 5.4: The estimated amount of code that can be reused from the original Skyview GCS application.

<table>
<thead>
<tr>
<th>Code reuse from SkyView GCS</th>
<th>Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UWP</td>
</tr>
<tr>
<td>Communication</td>
<td>89%</td>
</tr>
<tr>
<td>Visualizer (Map, 3D view)</td>
<td>69%</td>
</tr>
<tr>
<td>Displays</td>
<td>70%</td>
</tr>
<tr>
<td>Video component</td>
<td>61%</td>
</tr>
<tr>
<td>Core code</td>
<td>50%</td>
</tr>
<tr>
<td>Other</td>
<td>47%</td>
</tr>
<tr>
<td>Whole application</td>
<td>70%</td>
</tr>
</tbody>
</table>

As can be seen in Table 5.4, the Mono application could, based on the above estimation, reuse almost all of the code in a Mono application. However, there are a number of reasons why doing so is difficult. This will be further discussed in Section 6.2.4.

The UWP and Xamarin application have a quite similar expected reusability, differing with at most 8%. Typical components that could not be reused in the UWP and the Xamarin application were components from the Windows Forms library which is not supported by UWP and Xamarin.

Seeing that the communication component has high expected reusability is positive since that is the biggest and most relevant part of the original SkyView GCS software to reuse in a
new cross platform application. Converting the communication component should therefore be comparatively easy. This will be further discussed in Section 7.2.

5.5 Resource usage

In this section the results from the resource evaluation that was made on the different frameworks and platforms are presented. All applications were evaluated using the same sequence of application usage, described in detail in Section 4.7. First, graphs showing the CPU and memory usage for the different frameworks and platforms are presented and after that the results are summarized and compared to each other.

5.5.1 UWP

CPU usage

![CPU usage graph](image)

Figure 5.2: The CPU usage for the UWP application.

For the UWP application it is possible to see when the UAVs were moved and the lines were drawn in the CPU usage diagram in Figure 5.2. The Windows 10 Mobile seems to occasionally use more CPU than the Windows 10 PC. During the moving of the UAVs it used approximately 20% of the CPU while the Windows 10 PC only used about 5%.

Also, during the drawing of the lines the Windows 10 Mobile application used more CPU than the Windows 10 PC application with around 30% usage compared to the approximately 10% of the Windows 10 PC.

For the location changes, the Windows 10 PC application generally used more of the CPU than the Windows 10 Mobile application. For three out of the four location changes, the Windows 10 PC application used over 80% of the CPU while the Windows 10 Mobile application never reached more than 75%. On the other hand, the CPU usage of the Windows 10 Mobile application is slightly more spread out. This meant that in total they used their respective CPU approximately equally during this period of time.
5.5. Resource usage

Memory usage

![Memory usage diagram](image-url)

Figure 5.3: The memory usage for the UWP application.

When looking at the memory usage diagram in Figure 5.3, it can be seen that the Windows 10 PC application starts out using more memory than the Windows 10 Mobile application with around 90 MB memory used compared to 50 MB for the Windows 10 Mobile application.

During the moving of the UAVs and the drawing of the lines however, the memory usage of the Windows 10 Mobile application increased much more than the memory usage of the Windows 10 PC application. From 10 seconds to 50 seconds into the evaluation, the memory usage of the Windows 10 Mobile application increases with about 50 MB, leading to a similar memory usage as the Windows 10 PC application after all the lines had been drawn.

When the four location changes occur 57, 67, 77 and 87 seconds into the evaluation, the memory usage for both the Windows 10 PC and the Windows 10 Mobile application can be seen to increase slightly. The increase is on average 14 MB for the Windows 10 PC application and 8 MB for the Windows 10 Mobile application.
5.5. Resource usage

5.5.2 Xamarin

CPU usage

![CPU usage graph]

Figure 5.4: The CPU usage for the Xamarin application.

Just like in the UWP application, the Windows 10 Mobile application used more CPU during the moving of the UAVs and the drawing of the lines than the Windows 10 PC application. When comparing the Xamarin application to the UWP application during these evaluation events, it can be said that the Xamarin and the UWP application behaved very similar in general.

The Xamarin Android application had a bit slower start up than the Windows 10 PC and the Windows 10 mobile application which can be seen in Figure 5.4. During the movement of the UAVs and the drawing of the lines it behaved similarly to the Windows 10 PC application.

The big difference between the Xamarin and the UWP application can be seen in Figure 5.4 during the changing of the locations. In both the Windows 10 PC and the Windows 10 Mobile application, there are two spikes in CPU usage for each location change, excluding the third, while in the UWP application there is only one spike (see Figure 5.2). There is a simple explanation to this which was observed during the evaluation. The Xamarin application uses an animation when changing location while the UWP application instantly changes to the new location. The first spike in CPU usage was observed when the map was rapidly zoomed out. After that the map moved to the new location which can be observed as the short lowering of CPU usage and then it increases again as the map zoomed in on the new location. The reason why almost no second CPU spike can be observed at the third location change is that no zooming in on the map was necessary for that location (that location was the high altitude view of Bass strait, Australia). See Section 4.7 for more details about the resource test.

The two spikes in CPU usage does not occur in the graph for the Android application, even tough the same zooming in and out could be observed during the resource evaluation for the Android application. One reason for this could be differences between implementations in Google Maps and Bing Maps. Another could be differences between Windows and Android.
Not much will be said about the memory usage, (seen in Figure 5.5), of the Xamarin application except that the Windows 10 PC and the Windows 10 Mobile Xamarin application is very similar to the UWP application. The only real difference is during each of the four location changes were the Xamarin application used slightly more memory compared to the UWP application.

The big difference however is the Android application. The first real difference can be seen approximately 24 seconds into the evaluation as the memory usage suddenly drops from 20 MB down to 13 MB. At this point the Windows 10 PC and the Windows 10 Mobile application steadily increased their memory usage. During the location changes the memory usage is also different. As the location changes, the memory usage drops and then rises again. After all the location changes the application still used about the same memory as before the location changes with 19 MB after the location changes compared to just below 18 MB before the location changes.

Another thing that is worth noticing is that the Xamarin Android application used much less memory than the other platforms. This will be further discussed in Chapter 6.
5.5.3 Mono

CPU usage

For Mono, the Windows 10 PC and the Linux version of the application behaved similarly regarding CPU usage which can be seen in Figure 5.6. During the movement of the UAVs and the drawing of the lines on the map, the Windows 10 PC application used approximately 20-30% of the CPU while the Linux application used just below 40%. Worth noticing is that this is a higher CPU usage than the UWP and Xamarin application during these evaluation events.

When it comes to the location changes on the other hand the Mono application used less CPU compared to the other applications. The Mono application had a highest CPU usage peak of just 34% during the final change of location for the Windows 10 PC application. This is around 60 percentage points lower than the Windows 10 PC UWP and Xamarin application. The peak is similar in width to the peak in the UWP application and much more narrow than the peak in the Xamarin application. As discussed earlier in this section, this is due to the zooming in and out in the Xamarin application as the location of the map changes.
5.5. Resource usage

Memory usage

![Memory usage graph](memory_usage.png)

Figure 5.7: The memory usage for the Mono application. In the diagram Windows 10 PC is represented by the blue line and Linux is represented by the yellow line.

The memory usage of the Mono application stayed almost constant during the moving of the UAVs and the drawing of the lines as seen in Figure 5.7. For Windows 10 PC it goes from 39 MB after the start up to 40 MB after the drawing of the lines and for Linux it goes from 69 MB to 70 MB during the same period of time. These memory usages are lower than for the UWP and Xamarin application with the exception of the Xamarin Android application.

During the location changes the Windows 10 PC and Linux Mono application behave quite similarly. For the first and second location change, the memory usage increases significantly by between 4 and 17 MB. This is comparable to the average 14 MB of memory increase for the UWP Windows 10 PC application during a location change. During the third location change however, neither the Windows 10 PC or the Linux application increased their memory usage at all. The Linux application actually lowered the memory usage by approximately 0.5 MB. The third location was the high altitude view over Australia (see section 4.7 for details). Maybe the GreatMaps library, used for the Mono application, keeps a high altitude map of the earth cached at all times, which would lead to no extra memory being used during this location change. Another explanation could be that the application decided to remove some of the old map images at that point in time to save memory. For the last location change, the memory increased again, from 50 to 52 MB for the Windows 10 PC application and from 94 to 99 MB for the Linux application.
5.5. Resource usage

5.5.4 Summary

Before moving on to the discussion chapter a short summary of the results from the resource evaluation is useful.

![Average CPU usage](image)

Figure 5.8: The average CPU usage over the course of the evaluation for each method and platform evaluated.

The results from the CPU usage evaluations were fairly similar. Looking at Figures 5.2, 5.4 and 5.6 in each evaluation the moving of the UAVs, the drawing of the lines on the map and the location changes could be clearly distinguished from the idle periods. During the moving of the UAVs the CPU usage went up to values ranging between approximately 5% for the UWP application on the Windows 10 PC to approximately 35% for the Mono application running on Linux. The drawing of the lines raised the CPU usage to values from about 10% in the UWP, Windows 10 PC evaluation to about 40% for the Mono, Linux evaluation.

During the second part of the resource evaluation, with the four location changes, the Mono applications performed better, only peaking the CPU at about 35% while the UWP application reached peaks of over 90% CPU usage. The Xamarin application, excluding the Android version, also had peaks close to 100% CPU usage but this might have been due to the camera animation in Xamarin during location changes.

On average, the UWP Windows 10 PC application used the least amount of the CPU during the resource evaluation while the Windows 10 Mobile Xamarin application used the most which can be seen in Figure 5.8.
5.5. Resource usage

The results from the memory usage evaluation regarding the UWP and the Xamarin application when evaluated on the Windows 10 PC and the Windows 10 Mobile were fairly similar. The Xamarin application seems to use slightly more memory than the UWP application in general with the exception being the evaluation on the Android phone, were the memory usage were much lower than in any other test. That result will be further discussed in Section 6.2.5.

As seen in Figure 5.9, the Mono application used significantly less memory than the UWP and Xamarin applications (excluding the Android version), on both the Windows 10 PC and on Linux, with the Windows 10 PC using the least amount of memory (again excluding the Android application).

Worth noting is that although there were some differences between the applications, the memory usage was still fairly low in all of the evaluations, only reaching a maximum of 178 MB in the Xamarin, Windows 10 PC, evaluation.

Figure 5.9: The maximum memory usage for each method and platform evaluated.
This chapter discusses the methods used in this thesis and the results of this thesis in Section 6.1 and 6.2 respectively. After that the work in this thesis is placed in a wider context and some environmental and ethical aspects are discussed in Section 6.3.

6.1 Method

The general method used during this thesis was to develop the three applications to fulfill as many of the functional requirements from Section 2.4 as possible. In general this approach worked well, however some things could have been done differently.

The evaluation of code reuse from SkyView GCS in the UWP, Xamarin and Mono was performed after most of the implementation was complete. Looking at the high expected reusability values in Section 5.4 maybe more of the code from the SkyView GCS software could have been used to create the applications instead of creating them mostly from scratch. Especially in the Mono application more code could possibly have been reused which would have lead to a prototype application more similar to the original SkyView GCS software. However, as will be discussed later in this chapter, the values from this reusability evaluation might show a higher reuse than is actually possible.

The .NET Portability Analyzer that was used for evaluating the expected code reusability from SkyView GCS is quite limited. For example, it does not take into account how many times a missing API is used in the code. According to the team behind the tool, doing so would drastically increase the analysis time. This has very likely affected the results slightly.

In the work with the Xamarin application an iPhone with the Xamarin Live Player app was used for testing the application on iOS. Using this method for testing on iOS created several limitations due to limitations in the Xamarin Live Player app [29]. The big limitation for this thesis was that Custom Renderers was not supported. This prevented testing of both the map and the video component. Because of the limitations several of the functionalities could not be tested and the resource usage and the code reuse could not be evaluated for iOS.
To include a proper iOS implementation for the Xamarin application a Mac is necessary.

When testing the UWP and the Xamarin application on Windows 10 Mobile, an emulator was used. Using that approach had some drawbacks.

- **Time loss**
  The emulator could sometimes take several minutes to start which decreased the productivity when testing the applications. The emulator also suddenly stopped working and it took a long time to get it working again.

- **Validity of resource usage**
  The resource usage evaluation for the Windows 10 Mobile version of the UWP and the Xamarin application was made using an emulator. Since the Windows 10 Mobile emulator is only a software running on a normal Windows 10 PC the resource evaluation for Windows 10 Mobile might not be as valid as if a real device had been used.

- **Simulated touch input**
  The emulator had an option to switch between touch and mouse input. While this is certainly better than nothing, it is not as good as testing touch input using an actual touch screen.

When testing if the applications worked on different screen sizes some different approaches were used. On the Windows 10 PC and on Linux, the application window could be given a size on startup which could be used to simulate different screen sizes. It was also possible to resize the application while it was running. When testing on Windows 10 Mobile, two different emulators with different screen sizes were used. For the Xamarin Android and iOS applications only one device each was available to be used for testing. On these platforms the content of the application was instead given smaller space by adding a frame around it to simulate a smaller screen. Using actually differently sized screens would have been preferable over the approaches used in this thesis. On the other hand, both the Xamarin and the UWP application was tested both on a large computer screen and a small phone without any changes to the code.

During the resource test four different measurement tools were used as described in section 4.7. Because different tools might make their measurements differently, the results might differ. However, at least the comparison between the Windows part of the Xamarin application and the UWP application should be relevant since those comparisons was made using the same tool.

### 6.2 Results

The results obtained in this thesis were all focused on determining the suitability of UWP, Xamarin and Mono as tools for creating cross platform GCS applications. In this section the interesting and surprising parts of the results are discussed as well as the validity of some of the results.

#### 6.2.1 Prototype implementations

The three prototype applications were created with a similar visual design, using the same layout for displays and screens. The component that differed most between the application was the map component. As mentioned earlier in this thesis, GreatMaps was used as the map component for the Mono application. GreatMaps had some really nice features, such as the ability to manually cache map data to a local database, which the other map components
lacked. However, it also had some flaws such as the lack of an altitude property for the map markers, just like the Google Map component that was used in the Xamarin Android application, and the lack of support for adding map layers on top of the map.

### 6.2.2 Functional requirements

In general UWP, Xamarin and Mono managed to fulfill most of the functional requirements set by UAS Europe AB with the only exception being the possibility to import and show different layers on the map for the Mono application and the lack of 3D view for the Xamarin Android and the Mono application.

When testing the more detailed functional requirements for the Xamarin application, the most surprising result was regarding the lack of support for functional requirement number 14, loading GUI components dynamically from a .dll file, in the Xamarin UWP application. This is a serious limitation for several developers and a highly requested feature for UWP [30]. For UAS Europe AB, this functionality is important since it allows them to create a single GCS software and then allow their customers to customise the software with their own GUI displays.

An important thing to note about the features that were marked with a X (no support) in Table 5.1 and 5.2 is that just because the feature is marked with a X might not mean that it is completely impossible to accomplish for that platform. For example, the Xamarin Android application had no support for displaying markers with an altitude above the ground. This could possibly be implemented by offsetting the marker’s position on the map depending on the location and tilting of the ”camera” to make the marker look as if it is floating in the air. However, that would require several hours of work for something that was done by just setting the altitude property in the UWP application.

Since the Windows 10 PC and Windows 10 Mobile part of the Xamarin application are actually a UWP project, some of the results from the functional requirements are overlapping between UWP and Xamarin. If a feature is possible to implement in the UWP part of the Xamarin application it is going to be possible in a pure UWP application as well. This means that the results from the more detailed functional requirements, presented in Table 5.2 are applicable to a pure UWP applications as well. However, in several cases Xamarin does not provide code that works both for the UWP, the Android and the iOS projects and instead some platform specific code has to be written by the application developer.

### 6.2.3 Code reuse across platforms

The take away result from the evaluation of code reuse across the platforms for UWP, Xamarin and Mono is that almost all of the code for UWP and Mono could be used on both respective platforms without changes.

For the Xamarin application a large fraction of platform specific code had to be produced to get a functional map component, as just over a third of the code in the Xamarin application was shared between the UWP and the Android part of the application. Even though this might seem low, one can consider the alternative of creating a completely separate UWP application and an Android application using Java. This would allow for none of the code to be reused. Also, there is reason to believe that the amount of reusable code for the Xamarin application will rise as more and more business logic is added to it.
6.2.4 SkyView GCS code reuse

The analysis of potential code reuse of the SkyView GCS Code shows that the communication component had the highest proportion of reusable code, having over a 90% expected reusability for UWP, Xamarin and Mono. This is also the component that is the most important for UAS Europe AB to be able to reuse.

In the code reuse evaluation for the Mono application it looks as if almost all of SkyView GCS could be used in Mono. However, as mentioned in Section 6.1, the .NET Portability analyzer does not consider all aspects of reusability. In fact, some earlier work at UAS Europe AB shows that with some modifications parts of SkyView GCS can be compiled and executed using Mono but with large limitations in functionality. An employee at UAS Europe AB estimated that the actual values of reusable code was more likely around 70% for some of the components.

The most common type of non reusable code from the SkyView GCS application used the Windows Forms libraries. In general, it seems like using less components from the application model layer, and more components from the library layer (see Figure 3.1), leads to the code being more reusable. This makes sense since the base libraries are closer related to the CLI standard which both UWP, Xamarin and Mono implement.

6.2.5 Resource usage

The resource usage evaluation shows that none of the applications require a significant amount of memory to execute with the Xamarin Windows 10 PC application having the highest memory usage of 178 MB. Also, with the exception of the location changes, none of the applications seem to require very large amounts of computing power. UAS Europe AB’s requirement is that the applications should be able to run on new, high end devices and it is safe to say that doing so would not be a problem for any of the applications.

Some of the differences in the memory usage between the platforms have simple explanations. For example, the higher memory usage of the Windows 10 PC version of the UWP and the Xamarin application compared to the Windows 10 Mobile version can be explained partly by the fact that the Windows 10 PC used a much larger screen than the Windows 10 Mobile. Leading to more map images being loaded into the application and higher memory usage.

The extremely low memory usage of the Android application should be interpreted with great care because of the different tool used for obtaining the data, as discussed in the end of Section 6.1. However, the fact that the memory usage actually drops for some of the location changes is probably accurate. When manually testing the Android application it was discovered that it is extremely quick at removing old map data when the location changes.

The Mono application generally used less resources on Windows 10 than on Linux. These differences could have several explanations such as the fact that different tools were used when collecting the data and that the Linux application was tested using a Virtual Machine. In the study by Knittl [19], the execution times for seven different application was faster in Windows than in Linux for three of the applications.
6.3 The work in a wider context

Using UAVs to replace manned air vehicles has several implications. One of the positive aspects are the lower amount of fuel needed to operate a UAV compared to, for example, a manned helicopter. This should lead to a lower negative impact on the environment.

Using a small, light weight GCS, such as a tablet or a smartphone, as a replacement for a large GCS should lead to lower requirements during transportation and as an extension, lower environmental impacts. It could also lead to possibilities to conduct flights with drones in locations that were not reachable with a large and heavy GCS. This could include for example mountains during avalanches or volcano eruptions and forests during forest fires.

UAVs are used for several different purposes today. Use cases vary from for example recording movies to delivering packages. In today’s society there is a general concern about robots that replace human workers. This is discussed by both West [31] and Rotman [32]. This problem also applies to UAVs. A UAV can be vastly more efficient than a human in several aspects. For example at monitoring the water requirements of the different crops on a large crop field. By replacing say ten field working farmers with a UAV and an operator, nine work opportunities has been lost. Finding other uses for the now unemployed farmers can be a challenge for the society. Making the GCS hardware lighter and more portable might promote the UAV as a replacement for human workers even more.

The same concern can be applied to cross platform development. By eliminating the need to create one software for each platform, fewer employees might be needed to reach all the platforms required. On the other hand, a recent report by TEKSystems [33], suggests that there is a lack of talented programmers and developers in todays IT-market.

Making Ground Control Stations smaller and cheaper also makes them available to a larger number of people. One of the problems that has increased much in the last couple of years is the problem of protecting peoples privacy. There is an ongoing debate in Sweden if flying camera equipped UAVs without permission should be allowed or not and in 2016 it was decided that doing so should be forbidden. In 2017 however, the law was changed, making it legal to fly camera equipped UAVs again.
In this chapter the work in this thesis is concluded. In Section 7.1 the questions in the problem statement from Chapter 1 are answered. Lastly, the thesis ends with some suggestions on future work regarding both the Xamarin prototype and some suggestions on further comparisons for GCS softwares created using UWP, Xamarin and Mono in Section 7.2.

7.1 Answers to the questions in the problem statement

In this section the problem statement questions presented in Section 1.4 are answered.

1. **To what extent does UWP, Xamarin and Mono support the functionality needed for creating a cross platform Ground Control System software?**

   Based on the functional requirements provided by UAS Europe AB, which can be considered as fairly general, it can be said that UWP, Xamarin and Mono provide most of the functionality needed for creating a cross platform Ground Control System software. However, some shortcomings were discovered. UWP had simple built in components and libraries for almost all of the functionalities tested during this thesis. The only exception was the limitation on dynamically loading .dll assembly files. The Windows 10 PC and Windows 10 Mobile part of Xamarin has the same support for functionalities as the UWP application while the Android part lacks some important functionalities regarding displaying a 3D view of the map and the UAVs. Unfortunately the functionalities of the iOS part of the Xamarin application could not be fully evaluated, but testing and some research about Xamarin iOS suggests a result fairly similar to the Android application. Just like the Android application, Mono and GreatMaps lacks a way of showing the UAVs in a 3D environment. It also lacks functionality for displaying overlays on top of the map. The functionality of the Mono application was the same for both Windows 10 PC and Linux.

2. **In what ways does a Ground Control Software developed using UWP, Xamarin and Mono differ regarding CPU and memory usage?**

   The resource evaluation shows that there is no drastic difference between UWP, Xamarin and Mono regarding CPU and memory usage with the possible exception of the
memory usage in the Xamarin Android application. When looking at the Windows part of the Xamarin application it can be seen that Xamarin adds a small amount of overhead compared to the UWP application. Overall, the Mono application performed best concerning CPU and memory usage.

3. **Having a C# .NET Framework application, are there any typical segments of code that has to be rewritten in order to work on the different operating systems, using the different frameworks?**

The estimation on the amount of code reuse in Section 5.4 shows that the communication part of SkyView GCS should need the least amount of rewriting in order to work on the different operating systems, using the different frameworks. The main reason for the communication component being the most reusable is that it does not use any components from the Windows Forms library, which is not available in UWP and Xamarin. In general it can be said that using less components from the application model layer, and more components from the library layer (see Figure 3.1), leads to the code being more reusable. Since the application layer contains all GUI components, this means that non GUI code has a higher likelihood of being easily reusable than GUI code.

4. **How does the different frameworks compare regarding how much and what type of platform specific code that has to be written to create a GCS application?**

The amount of platform specific code required varies considerably between the platforms. In the UWP application, no platform specific code had to be written at all, while in the Xamarin application more than half of the total amount of code was platform specific. The largest portion of this code was related to the map and the UAV icons displayed on the map. A small amount of code is also necessary to make a simple video component that could be used both on the Windows 10 platforms and on Android. The Mono application required a negligible amount of platform specific code for the video component and no platform specific code for the map and the map icons.

### 7.2 Future work

The subject of cross platform development has several opportunities for further work. In this section some suggestions related to the prototypes created in this thesis as well as further comparisons between UWP, Xamarin and Mono are given.

#### 7.2.1 Xamarin prototype

The Xamarin application is still a prototype. To make it into a proper, production-grade application, much more work is needed. One interesting extension to the application would be to use the existing SkyView GCS communication component as a basis for communication in the Xamarin application. In fact, the work of converting the communication component for Xamarin has already started at UAS Europe AB.

Another area were much more work can be done is in the iOS part of the Xamarin application. Because of limitations in the Xamarin Live Player app the iOS part of the Xamarin application is much less developed than the UWP and the Android parts. Using a Mac for testing, the Xamarin application could be extended with more functionality for iOS.

The Xamarin Android prototype was in this thesis only tested on a single Android smartphone. It would be preferable to ensure the applications functionality on other devices as well, such as other Android smartphones but also Android tablets which is possibly the most important platform for UAS Europe AB to support. This could be done by either buying additional physical devices or by using the Xamarin test cloud which is a paid cloud service that provide developers access to physical devices to test their application on.
7.2. Future work

7.2.2 Further comparison of UWP, Xamarin and Mono

In this thesis the resource evaluation only included CPU and memory usage. This evaluation could be extended to also include other metrics such as execution time, utilization of graphics processing unit and storage space requirements just to name a few. By doing so, possible limitations could be discovered, which might affect the choice of framework for a project. It would also be interesting to redo the comparisons using real Windows 10 Mobile phones instead of emulators.

As discussed in Section 6.2.5, the memory usage of the Android application was very low. This could be further studied by either trying to find or develop a single tool that can measure memory usage for all the applications. Another approach would be to find a second measurement tool for the Android application and then compare the measurements to the results from the measurements by Android Monitor which was used in this thesis.

Another interesting comparison could be made using the Mono application. In the paper by Knittl [19], the execution time of an application using Mono and traditional .NET Framework is compared. It was found that in all of the tests, the .NET Framework application performed better than the Mono application. In this thesis the Mono application was only tested using Mono on the Windows 10 PC and the Linux Virtual Machine. However, it should be possible to run the application using the traditional .NET Framework on the Windows 10 PC and see how that affects resource usage and functionalities.
Bibliography


[13] CP Rahul Raj and Seshu Babu Tolety. “A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach”. In: India Conference (INDICON), 2012 Annual IEEE. IEEE. 2012, pp. 625–629. DOI: 10.1109/INDICON.2012.6420693


