Image Organizer

by
Adam Strupczewski
SID: 2394167
# Table of contents

Declaration of Originality ........................................................................................................ 4  
Copyright Acknowledgement .................................................................................................. 4  
Acknowledgements .................................................................................................................. 5  
Abstract .................................................................................................................................... 6  
1. Introduction .......................................................................................................................... 7  
   1.1. Project Background ......................................................................................................... 7  
   1.2. Project Objectives .......................................................................................................... 8  
2. Project Management ............................................................................................................. 9  
   2.1. Project Requirements and Deliverables ........................................................................ 9  
       2.1.1. Coventry University requirements ......................................................................... 9  
       2.1.2. Client’s requirements ............................................................................................ 9  
       2.1.3. Project deliverables ............................................................................................... 10  
   2.2. Potential Risk Management ......................................................................................... 11  
   2.3. Project Stakeholders ..................................................................................................... 12  
   2.4. Project Schedule and Task Planning ............................................................................ 12  
3. Background Research .......................................................................................................... 15  
   3.1. Topic Overview ............................................................................................................. 15  
   3.2. Similar Software Analysis ............................................................................................. 16  
       3.2.1. General overview of image sorting software ......................................................... 16  
       3.2.2. Caliph & Emir ....................................................................................................... 17  
       3.2.3. The PicSOM system .............................................................................................. 19  
       3.2.4. ImageSorter .......................................................................................................... 20  
   3.3. Client’s Software and Hardware Analysis ....................................................................... 22  
   3.4. Tool and Technology Choice ....................................................................................... 23  
   3.5. Development Methodology ......................................................................................... 25  
4. Design .................................................................................................................................... 27  
   4.1. Concept of Similarity Sorting ....................................................................................... 27  
   4.2. Image Feature Description ............................................................................................ 28  
       4.2.1 Color Layout Descriptor .......................................................................................... 28  
       4.2.2 Scalable Color Descriptor ....................................................................................... 31  
   4.3. Similarity Sorting Algorithm ....................................................................................... 34  
       4.3.1 The Incremental Self-Organising Map Algorithm .................................................. 34  
       4.3.2 The Batch Self-Organising Map Algorithm ............................................................. 36  
   4.4. Keyword Filtering ......................................................................................................... 38
Appendix B

References

Bibliography

Appendix A – Project Brief

Appendix B – Cooperation with Client

B-1. Initial Agreement
B-2. Feedback for Prototype 1 .................................................................................................................................................. 82
B-3. Feedback for Prototype 2 .................................................................................................................................................. 84
B-4. Evaluation of Final Product .................................................................................................................................................. 86
Appendix C – Additional Materials on CD .................................................................................................................................. 87
Declaration of Originality

This project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged within the main report to an item in the References or Bibliography lists.

I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

Copyright Acknowledgement

I acknowledge that the copyright of this project and report belongs to Coventry University.
Acknowledgements

I am most grateful to my client, Piotr Fetraś, for very good cooperation and a lot of time that he devoted to help me improve my project.

Special thanks to my supervisor, Ph.D. Chris Peters, who provided me with numerous tips and useful advice.

Many thanks also to Ph.D. Grzegorz Galiński, my former multimedia lecturer, who always offered help when I needed it.

Last but not least, I wish to thank my fellow Erasmus friends, Piotr Stępień and Michał Płatek, for help with testing my application.
Abstract

This report describes an application intended for sorting images - Image Organizer. The application is capable of sorting images in a number of ways, most notably however it enables image sorting according to their visual similarity.

The aim of this document is to present all stages of the application’s development. First it covers background research and requirement gathering. Then it moves on to explain the design and implementation process. After that it focuses on testing and evaluation of the application.

A user manual explaining how to use the application is on the accompanying CD as a separate document and is not part of this report.
1. Introduction

1.1. Project Background

In the contemporary world collections of digital images are growing ever more quickly. This obviously leads to the need of efficient sorting and organisation. There are numerous schemes of sorting images according to name, size or metadata describing the image. However, with large databases of images these ways of sorting may not be enough. The user might want to find an image that is similar to a certain image that they specify. Systems that enable this - Content Based Image Retrieval (CBIR) systems – are a fairly new concept and are constantly being improved.

CBIR systems offer an innovative, powerful approach to image searching and are certainly useful for quickly finding what one is looking for. But what if no query image is available? In that case such a system will not work, as there will be nothing to compare images from the digital library with.

People have tried to think of a way to overcome this drawback. The result has turned out to be a completely new concept - sorting images according to their content, and more specifically visual similarity. Thus, even with no query image available, the user is able to quickly find what they are looking for (Barthel 2008).

Systems that focus on image visual similarity sorting are a new idea, and it would probably be fair to say that the ones existing at present are merely prototype versions. Furthermore, very few lay people know of their existence at all. The author only found out about this concept during a lecture on multimedia, which they had attended the previous year in Poland. This topic has since been discussed with a number of people, arousing a lot of interest. A certain photographer, Mr Piotr Fetraś, expressed the desire to try this new way of sorting with his own image collections. When it was later realised that this new sorting scheme could be implemented in an application being part of a Bachelor Degree project, Mr Fetraś was once again contacted and he agreed to be the client for such an application.
1.2. Project Objectives

In order to successfully complete any project, its aims need to be clearly identified and well thought over before work on the project is started. This way particular stages can be appropriately planned, thus avoiding any unnecessary time waste. The project includes the following objectives.

1. Requirement gathering

This stage consists in collecting information from the client and possibly other intended users regarding the final product that they wish to have. This information should be the foundation for developing prototype versions and eventually the final version of the product.

2. Similar product analysis

It would be a waste of time to develop something that is already available on the market. The created product needs to be in some way innovative, showing a new approach or a new way of solving a problem. Therefore, a detailed analysis of software available (in case of programming) should be performed, and any related programs and already implemented ideas should be made use of. This way work may be focused on new, innovative features, instead of wasting effort on things that have already been done by someone else. An analysis of related work may also help to avoid mistakes that others have made.

3. Background research and justified technology choice

Before actually starting work, it first has to be specified what tools will be needed. When talking about software, it needs to be established which programming languages and potentially libraries will be most suitable. All other debatable issues should also best be at least partially solved at the research stage. This allow avoiding unexpected complications and can often lead to more efficient problem solving.

4. Prototype design and implementation

Once the requirements are gathered and the technical issues solved, it is time to design the first prototype version of the product and create it. This is a major step towards completing the product and usually requires most effort on part of the developer, so it needs to be well planned in terms of time management.

5. Feedback gathering and prototype improvement

After the first prototype version is created, it can be shown to the client and other intended users. They can then provide feedback regarding their impression and vision as to what the final version should look like. This feedback can be used to improve the product, so that it matches the needs better. Usually this is an iterative process and several prototype versions are created before a final version of a product is created. Each consecutive prototype version should be an improvement of the previous one.

6. Final version production and evaluation

After several contacts with the client and a number of modifications in the product prototypes, a final version matching the client’s needs can be created. This final release ought to be evaluated and areas of further improvement for future work should be identified.

Each of the above stages will be covered in much greater detail later on in this report.
2. Project Management

Project management can be defined as

“the application of knowledge, skills, tools and techniques to a broad range of activities in order to meet the requirements of a particular project”

(Project Management Institute 2000)

As stated in the above definition, by a very competent organisation in this matter, project management requires a wide variety of skills and activities to make it successful. Many people neglect the need to manage a project well, but a lot of them later regret the decision when something unexpectedly goes wrong, and catches them completely unprepared.

In general, it can be said that good project management has to include analysis in the following areas:

- Project requirements and deliverables
- Potential risk management
- Interaction with project stakeholders
- Task planning and time management

These areas will now be shortly discussed.

2.1. Project Requirements and Deliverables

The project is done for two different groups of people. One is the client and other potential users of the product, while the other includes the assessors at Coventry University – the supervisor and the second assessor. Each of these parties requires something slightly different, so it is best to divide the overall project requirements and deliverables into two groups.

2.1.1. Coventry University requirements

As to requirements set out by Coventry University – within the 314 CR Creative Project module, they are as follows:

- Completion and delivery of project preference form within deadline
- Completion and delivery of project brief within deadline
- Cooperation with project supervisor
- Oral presentation
- Optional product demonstration
- Completion and delivery of final report and product within deadline
- No plagiarism

2.1.2. Client’s requirements

Requirements gathered from the client are something different, as they specify exactly what the final product should be like. These requirements were gathered throughout the whole
project development process, though most were listed in the initial agreement which was signed after the first interview with the client (Appendix B: B-1). The main aims for the final product are as follows:

- The final product should be an application suitable for the Windows operating system (portability for Linux is not required)
- The application should be capable of sorting and displaying large sets of images according to their visual similarity
- The required image format that the application should support is only JPEG, but support of other formats would be greatly appreciated
- The application should be capable of displaying basic information about any chosen image, such as location, size, resolution etc.
- The application should offer the possibility of magnifying any chosen region of images to see it in greater detail
- The application should enable easy selection of any chosen image or group of images and allow basic operations, such as moving, deleting or editing on any selected image or group of images
- The application should further enable filtering images by keywords, and should provide a way to edit keywords describing each image

It is important to note that these were the general goals established during the first interview with the client. Many were more precisely restated in form of feedback that the client provided after testing prototype versions of the product. All this will be covered in more detail in chapter 6.

2.1.3. Project deliverables

Based on the requirements imposed by the client and the university assessment, it is possible to discern the project’s deliverables. The deliverables can be classified as below:

1. A completed final version of the product, which is to be an application satisfying all conditions mentioned in point 2.1.2. The application should be equipped with a user-friendly graphical interface and should also fulfil any extra requirements that the client provides in form of feedback.
2. A minimum hardware specification and user manual for the application.
3. A project brief to be handed in at an early stage of the project's development.
4. A report describing the development process. The report should include an overview of:
   - cooperation with the client
   - background research regarding similar products and the intended user’s needs
   - project management and task planning
   - algorithms used
   - problem solving
   - product evaluation
   - optionally an indication of areas which could be improved in the future
2.2. Potential Risk Management

To prepare for problems and unexpected difficulties it is a very good practice to carry out a risk analysis. A risk analysis classifying different kinds of risks, the probability of their occurrence and their importance in terms of influencing the project is presented below.

<table>
<thead>
<tr>
<th>Type of risk</th>
<th>Probability (L-M-H-VH)</th>
<th>Significance (L-M-H-VH)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Risks:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data loss</td>
<td>L</td>
<td>VH</td>
</tr>
<tr>
<td>Unexpected personal problems</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td><strong>Initial risks:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inability to find client</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Wrong choice of project in terms of possessed skills and knowledge</td>
<td>L</td>
<td>VH</td>
</tr>
<tr>
<td>Wrong choice of project in terms of the possibility to complete it in the given time or at all</td>
<td>L</td>
<td>VH</td>
</tr>
<tr>
<td><strong>Research related risks:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inability to gather precise requirements from client</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Inability to find sufficient information about the topic of the project</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td><strong>Design related risks:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wrong technology choice</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>Wrong tools choice</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>Wrong choice of approach to the problem</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>Neglecting the need to do a good design</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td><strong>Implementation related risks:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insufficient knowledge of tools or methodologies used</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>Finding flaws in the design</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Encountering insolvable problems</td>
<td>L</td>
<td>VH</td>
</tr>
<tr>
<td><strong>Testing related risks:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discovering bugs in product design</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>Discovering unfixable bugs in the final version of the product</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td><strong>Report writing related risks:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starting work on the report too late, without enough time left to do it properly and deal with problems</td>
<td>VH</td>
<td>H</td>
</tr>
<tr>
<td>Unintentional plagiarism</td>
<td>L</td>
<td>H</td>
</tr>
</tbody>
</table>

Table 2.2-1 Risk assessment

Obviously all the risks as a whole pose a serious threat to the successful completion of the project. Therefore, all that is possible should be done to ensure that the impact of these potential risks will be held at a minimum.

The following actions were taken to achieve this:
- Careful choice of client in terms of cooperation abilities
- Exhaustive background research into the technologies and algorithms necessary for the project completion
- Clear establishment of goals for the project
2.3. Project Stakeholders

Project stakeholders are all the people who have any interest in the project. If they are identified early on in the development process, they can be ‘managed’ better, which is generally beneficial for the project.

The stakeholders identified for the project are listed below.

1. The client
The client is the person who is most interested in the successful completion of the project. It is them who want the final product, and it is them who provide most of the requirements for the product. The client also provides feedback regarding prototype versions of the product and it lies in their interest to give any useful tips regarding the product design. Therefore, the client’s interest in the project and the influence over the final outcome are both high. This leads to the conclusion that the client should be closely cooperated with and kept satisfied.

2. The supervisor
The supervisor is the person who assesses the final outcome of work on the project. It is also their job to provide assistance and give tips throughout the development process.

In general the supervisor’s interest in the project is also high, though perhaps not as high as the client’s. The supervisor’s power in form of assessment however cannot be underestimated. Therefore, good liaison with the project supervisor and ensuring that all requests of the supervisor are met is recommended and lies in the developer’s personal interest.

3. The second assessor
The second assessor is supposed to help the project supervisor with the assessment of the project. Their influence is significant, though their interest in the project itself might not be too high. Therefore, an interesting and attractive way of presenting the project outcome might be helpful to grip their attention and encourage them to award higher marks.

2.4. Project Schedule and Task Planning

Detailed task planning is an essential part of every project. Good planning minimizes the risk of failure and ensures that each stage of the project is completed at the right moment. Sometimes certain parts of the project have to be completed one after the other and good planning can help determine the right order of doing things, so as not to waste time unnecessarily.

After the requirements were gathered and the first stages of background research were completed, it was possible to break up the whole project into several categories, each including a number of tasks. This categorization was done in order to clearly identify all the
different stages which would require attention and effort. Thus a good, detailed plan of all awaiting tasks could be prepared more efficiently.

The general categorization of all tasks is presented below.

![Image Organizer Project]

Based on this categorization a more detailed work chart could be prepared. All tasks were planned in logical order, most with exact start and finish dates. Also most meetings with the client were arranged and planned at that stage. The whole schedule can be well depicted on a Gantt Chart as shown on the next page (Figure 2.4-2).

The Gantt Chart assumes a 7-day per week work schedule, as when working at home most of the time there isn’t much difference between weekdays and weekends. All tasks have allocated time slots for their completion. It is assumed that work starts on the very first day that each time slot covers. In this case all time slots include a margin for problem solving, should any difficulties occur. If for example a task is scheduled for 6 days, it would probably require just 3 days to complete it, unless something unexpected arises. Because of this following the task schedule was feasible, and given regularity in working on the project - not even difficult.

It is worth noting that some stages marked on the chart were planned at the very beginning of work on the project (for instance research and design tasks), whereas others were planned throughout the development process (such as prototype improvement – this could only be done after the client provided feedback). Nonetheless, planning tasks in advance has been found very useful and made work more efficient.
<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>University related work</td>
<td>147 days</td>
<td>Thu 06-10-02</td>
<td>Fri 09-04-24</td>
</tr>
<tr>
<td>Supervisor allocation</td>
<td>16 days</td>
<td>Thu 06-10-02</td>
<td>Wed 06-10-22</td>
</tr>
<tr>
<td>Project brief</td>
<td>27 days</td>
<td>Wed 06-10-22</td>
<td>Fri 08-11-28</td>
</tr>
<tr>
<td>Brief preparation</td>
<td>27 days</td>
<td>Wed 06-10-22</td>
<td>Thu 08-11-27</td>
</tr>
<tr>
<td>Brief submission</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project presentation</td>
<td>10 days</td>
<td>Thu 09-02-26</td>
<td>Thu 09-03-12</td>
</tr>
<tr>
<td>Presentation preparation</td>
<td>10 days</td>
<td>Thu 09-02-26</td>
<td>Wed 09-03-11</td>
</tr>
<tr>
<td>Presentation delivery</td>
<td>0 days</td>
<td>Thu 09-03-12</td>
<td>Thu 09-03-12</td>
</tr>
<tr>
<td>Final report</td>
<td>28 days</td>
<td>Fri 09-03-20</td>
<td>Fri 09-04-24</td>
</tr>
<tr>
<td>Report preparation</td>
<td>25 days</td>
<td>Fri 09-03-20</td>
<td>Thu 09-04-23</td>
</tr>
<tr>
<td>Report submission</td>
<td>0 days</td>
<td>Fri 09-04-24</td>
<td></td>
</tr>
<tr>
<td>Development work</td>
<td>127 days</td>
<td>Wed 06-10-01</td>
<td>Tue 08-03-31</td>
</tr>
<tr>
<td>Contacts with client</td>
<td>114 days</td>
<td>Mon 06-10-20</td>
<td>Fri 08-03-27</td>
</tr>
<tr>
<td>First skype interview and requirements gathering</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial agreement preparation (in Warsaw)</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feedback acquisition after presenting prototype 1</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feedback acquisition after presenting prototype 2</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feedback acquisition after final version</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background research</td>
<td>10 days</td>
<td>Tue 06-10-01</td>
<td>Tue 08-10-26</td>
</tr>
<tr>
<td>Gathering literature</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Similar system analysis</td>
<td>10 days</td>
<td>Tue 06-10-07</td>
<td>Tue 08-10-19</td>
</tr>
<tr>
<td>Technology and tool choice</td>
<td>5 days</td>
<td>Tue 06-10-14</td>
<td>Tue 08-10-19</td>
</tr>
<tr>
<td>QT installation and learning</td>
<td>5 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>20 days</td>
<td>Wed 06-10-29</td>
<td>Tue 08-11-26</td>
</tr>
<tr>
<td>MPEG-7 descriptor choice</td>
<td>6 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorting algorithm choice</td>
<td>6 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Similar sorting design</td>
<td>3 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword filtering design</td>
<td>4 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>General program architecture design</td>
<td>16 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>58 days</td>
<td>Wed 06-11-26</td>
<td>Sun 09-03-15</td>
</tr>
<tr>
<td>Basic GUI creation</td>
<td>6 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image loading and display</td>
<td>5 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color layout descriptor implementation</td>
<td>5 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalable color descriptor implementation</td>
<td>6 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOM algorithm implementation</td>
<td>6 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Batch-SOM algorithm implementation</td>
<td>5 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm testing and final choice</td>
<td>5 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation of keyword utilities in application</td>
<td>6 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saving keywords into graphic file implementation</td>
<td>15 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final improvements</td>
<td>5 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final prototype ready</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype improvement and testing</td>
<td>23 days</td>
<td>Wed 06-02-18</td>
<td>Sun 09-03-22</td>
</tr>
<tr>
<td>Improvement of prototype 1</td>
<td>6 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improvement of prototype 2</td>
<td>16 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final tests</td>
<td>5 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final version of application ready</td>
<td>0 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual Preparation</td>
<td>8 days</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Background Research

3.1. Topic Overview

As mentioned in the introductory chapter, the application that is the aim of the project is meant to be capable of sorting images. One might expect to easily find hundreds of programs that perform picture sorting, and rightly so. However, a great majority of these will enable sorting only in a most basic approach – one dimensional sorting of images according to certain external metadata saved along with the image – be it the image name, size, resolution or keywords describing it.

Such a simple approach is very useful and effective, but sometimes is not enough. It is quite easy to imagine the need to find a certain image that one knows what it looks like, but cannot provide any metadata data about it. To satisfy this need, Content Based Image Retrieval (CBIR) techniques have been developed. These techniques require that the user provides a query image which is then compared with all images in a digital library. After all comparisons are finished, the nearest matches are displayed. However, the obvious disadvantage of this approach is that a query image is necessary to make it work.

The aim of the project was to implement a completely new notion of image organising, so far tested by very few people. The new idea combines the advantages of classical sorting and CBIR techniques. It involves sorting images not by metadata, but by their visual similarity. Therefore, any image that is known what it looks like can quickly be found, and without the need to provide a query image.

The new, visual similarity concept of sorting has only been introduced a couple of years ago, and is still at the stage of prototyping. Most research regarding this subject has been carried out by Prof. Dr. Kai-Uwe Barthel (Barthel et al. 2005, Barthel 2008). Other work in this area has never succeeded in actually producing a working program. All systems that are somehow related to visual similarity sorting will be analysed in the next section.
3.2. Similar Software Analysis

As mentioned before, there are numerous programs available in shops and on the internet, which allow image sorting to be performed. In order not to reinvent something that is already common knowledge, it is necessary to perform an analysis of all software that may include useful solutions for this project. Apart from software that has something to do with Self-Organising Maps or MPEG-7 descriptors, it is important to also investigate other programs meant for image sorting, as some ideas implemented in them might turn out to be useful.

3.2.1. General overview of image sorting software

Programs that enable sorting images according to basic file data (name, size etc) or added metadata such as keywords are available in great abundance. The most popular are SortPics, Photo Sorter Platinum and Sort Photos. Photo Sorter Platinum and Sort Photos are focused on automatic sorting of all images on one’s hard drive according to chosen features - it is possible to choose image size, date taken etc. (Sort Photos 2009, Photo Sorter 2009)

SortPics is a program more focused on adding tags to images and classifying them according to keywords and categories. The main panel of SortPics looks as presented below (SortPics 2009).

![Figure 3.2.1-1 SortPics](image)

As it can be seen, the program provides an easy way of viewing all folders on the computer in the tree-view present on the left. Loading any image is simply done by clicking with the mouse.

Another program that has to be mentioned when talking about image sorting is Picasa (Google 2009). It keeps record of all image files on the computer and enables quick and easy viewing of many images at a time. It is generally intended for making copying of images to a
web service easier, but it works smooth and also makes moving multiple images to new folders on the hard drive very easy. A screenshot of its main panel is shown below:

![Figure 3.2.1-2 Picasa 3](image)

Picasa also enables rating images, opening them for editing and hiding chosen ones from the display. Another very useful feature is the possibility to choose a custom zoom level – it can be specified whether to display smaller image miniatures and thus fit more on a single screen, or whether to view images in more detail.

The general conclusion is that although all of these programs perform something different than is aimed to be achieved by the developed application – Image Organizer, several ideas implemented in the tested programs should be considered, as they may potentially be useful. These ideas include:

- Clear and easy to navigate tree-like view of all directories on the computer
- Choosing a custom zoom level of the displayed images
- Selecting multiple images from those selected by holding the SHIFT key, pressing the mouse and dragging it, later the possibility to do things with the selected images via a context menu that appears when the right mouse key is pressed
- The possibility of editing text describing the image in form of free-text keywords as well as predefined categories
- The possibility of sorting images by different criteria: name, size, date created, resolution etc.

### 3.2.2. Caliph & Emir

Caliph and Emir are two programs that are closely related – the first enables editing data describing an image, while the second enables image retrieval according to chosen criteria (Lux 2007).

Caliph (Common And Lightweight Photo annotation application) is a really powerful tool for describing an image. It enables doing it in several different ways:
1. Editing structured text and free text comments, as well as providing metadata about the image
2. Editing semantic relations that describe the image
3. Saving shapes that appear in the image
4. Saving four MPEG-7 descriptors describing the image (EdgeHistogram, DominantColor, ColorLayout, ScalableColor)

The most innovative of all these possibilities is the option to edit semantic relations. It allows creating objects – people, places, events etc, and determining how they depend on each other. This can later be used for retrieval of images that match specified semantic relations. The main panel for editing these relations in Caliph, along with an exemplary relation network, is shown below:

![Figure 3.2.2-1 Caliph](image)

Caliph allows saving all image annotations in form of MPEG-7 xml files. These can be later used by Emir (Experimental Meta data based Image Retrieval application) to find specific images.

As it could be expected, Emir allows searching for images in many ways – matching all that could have been edited with Caliph. Therefore, it enables searching for images according to both free text and structured comments, according to semantic relations and according to any query images, that are later compared in terms of the four previously mentioned MPEG-7 descriptors with all images in the digital library.

However, the most interesting feature of Emir from the point of view of this project, is the ability to display all the found images matching the given criteria in an organized way – considering the visual similarity of the images. It is possible to choose one descriptor for this visualization, and all images will be displayed on a 2-D surface in such a way, that more
similar images will be closer to each other, whereas images having less in common will be further apart. The algorithm that Emir uses to achieve this is the FastMap algorithm. Unfortunately, this type of visualization technique places images in a continuous 2-D space, which inevitably leads to many images overlapping each other. An example of this for a relatively small number of images (less than 100) is shown below:

![Image Organizer](image-url)

Figure 3.2.2-2 Emir

This is a similar result to what the designed Image Sorter is aiming for. However, overlapping of images has to be somehow solved, so the FastMap algorithm needs to be more closely investigated. As to other conclusions after testing Caliph and Emir, the following ideas are deemed interesting and worth considering in Image Organizer:

- Describing images with the use of MPEG-7 descriptors
- Loading and saving annotations in form of MPEG-7 XML files
- Separating filtering and organised image display – the possibility of choosing a subset of images from the whole library to be displayed as visually sorted

### 3.2.3. The PicSOM system

The PicSOM system implements Content-Based Image Retrieval, so in general it needs a query image as the input, and returns a set of images as the output, which is a result of comparisons between the query image and all images in the digital library. However, the system is innovative compared to the classic approach in a number of ways.

First of all, PicSOM is designed to collect information from the user after performing a search in order to determine the accuracy of the results. This is called Relevance Feedback (RF) gathering (Laaksonen et al. 2002). It is aimed to solve the problem of matching low-level feature descriptors with high-level human object perception. In each particular case, some descriptors might express the aim of the search more closely, while others will not be
relevant at all. When feedback is provided after each search, the program can adjust weights of different descriptors to each particular image query, so that the descriptors match the user’s preferences more closely and thus give better search results. Therefore, searching with PicSOM is an iterative process, where the system is capable of learning from the user.

The other novelty of PicSOM, which is more related to this project, lies in the way of finding similar images in the database. This retrieval is done with the use of several parallel SOMs - Self Organized Maps. Let us look into this topic a little more closely (Laaksonen et al. 2002).

First of all, before any searching is performed, PicSOM trains a number of 2-D SOMs with different sets of images. These SOMs are structures which map sets of images in a two dimensional space in such a way, that visually similar images appear close to each other. Depending on the images that were used to train the SOM, and the chosen image features (descriptors), each of these maps will be different. When the search is initiated by the user, all of these different SOMs are used to find similar images to the query. Then, after gathering feedback from the user, the system gets to know which images were chosen as accurate matches and can redefine the importance of each particular SOM for the search. This way, in each consecutive search better results can be achieved.

The most interesting thing about the created maps by the PicSOM system is their special form. They are not just ordinary SOMs, but hierarchic 4-level Tree-Structured SOMs. Thanks to this the topmost SOM has small dimensions, and each of its nodes has a new SOM under it, that contains a lot more nodes. This pattern continues to form four separate SOM layers. The main gain of such an approach is the much higher search efficiency – the time complexity to find the most similar image to the query is \( \Theta(\log n) \) instead of \( \Theta(n) \) as in the classic SOM, where \( n \) is the number of nodes in the SOM (so roughly the number of images). For more information see Laaksonen et al. 2002.

To recapitulate, the PicSOM system introduces several interesting ideas, that may potentially be useful when creating Image Organizer. The following concepts should be considered:

- Use of MPEG-7 descriptors as image description
- Use of the Self-Organising Map algorithm to sort images in two dimensions
- Training SOMs before the program is actually run and saving those SOMs
- Idea of hierarchical SOM, though it may be difficult to implement because of necessity to display sorted images in Image Organizer
- Use of different image features to sort different images

3.2.4. ImageSorter

ImageSorter is a program developed by Kai-Uwe Barthel for automatic Image Sorting (Barthel et al. 2005, University of Applied Sciences – Berlin 2008). It uses MPEG-7 descriptors and Self-Organising Maps to visualize all images on a surface or sphere, according to the image content. It performs this visualization quite efficiently and sets a good example of how image similarity sorting can be performed. It also has a user-friendly gui and is quite easy to navigate. A screenshot of the v3.0 BETA 3 version (University of Applied Sciences – Berlin 2008) is shown below.
Figure 3.2.4-1 ImageSorter while displaying a sorted set of images on the surface of a sphere

ImageSorter has a very good main panel structure. It includes a tree-like view of the directories on the computer, which makes it easy to load images from any selected folder or folders. It also enables very handy zooming with the mouse wheel and preview of any image that the mouse hovers above. It also allows the selection of a subset from the displayed images and performing some basic operations on it. Unfortunately, there are a number of things that limit its functionality. Firstly, it doesn't allow adding any textual description to images, and so also doesn't enable filtering images according to it. Secondly, once images have been added, it is necessary to restart all sorting when a new image is added. As to sorting speeds, they are entirely satisfactory with a number of images not exceeding several thousand, but for more the sorting takes a long time.

Despite all the ImageSorter’s imperfections, it still is a very good tool and there are numerous ideas implemented in it which can be made use of within this project. They are as follows:

- The concept of using MPEG-7 descriptors and Self Organizing Maps for image similarity sorting
- The GUI layout – easy to navigate
- The option of zooming with the mouse wheel
- The possibility of hiding selected images and resorting those remaining
- The option of choosing a custom background colour

All mentioned ideas in points 3.2.1 – 3.2.4 have been considered regarding the use in Image Organizer. Those deemed worthy of implementation have been tried and tested. More information regarding this is provided in chapters 4 and 5.
3.3. Client’s Software and Hardware Analysis

In order to create an application that will be useful for the client, it has to be designed for the software that they are using, and the application’s minimum hardware requirements cannot exceed the hardware that is at the client’s disposal. That is why analysing the client’s hardware and software, even if it is a straightforward matter, is absolutely crucial.

As to the software, the client currently uses Windows Vista Home Premium. Furthermore, when discussing the future, the client intends to continue on using the Windows operating system. This means that the application needs to be compatible with Windows, but necessarily with other systems. Nonetheless, if the application was designed in a system-independent manner and was portable to Linux, or at least require few changes to make it portable, that would be an additional merit.

The hardware is an entirely different issue altogether. Luckily for the developer, Mr Piotr Fetraś has fairly new and reliably equipment at his disposal. The computer that the software will most likely be used for includes, among others, the following elements:

- Intel Core 2 Duo CPU, 3.2 GHz
- 4 GB of RAM
- 320 GB hard drive

As to the hard drive, it may potentially offer up to gigabytes of space, even though this much will certainly not be required. However, it will certainly be appreciated if the application along with the necessary components take up as little space as possible.

The processor and RAM are key factors determining the speed of the application. In case of a rather small program such as the one aimed to be developed, the general functionality shouldn’t really depend on these elements. However, certain complex operations, such as image loading or image sorting may require a lot of computational power as well as RAM, therefore they should carefully be considered to perform satisfactorily on the client’s computer. All these issues will be discussed in greater detail in chapter 5.
3.4. Tool and Technology Choice

In order to start programming, one has to decide about several things before the very first line of code is written. The first decision regards the programming language. The second regards the compiler. Another thing that often has to be decided at the very beginning is which optional libraries will be used.

As to the programming language, there were several choices that could have been made. The more popular choices could have been Delphi, C/C++, Java, Perl, Python or one of Microsoft’s languages such as C# or Visual Basic. The limitation of choices from all of these was mainly determined by the developer’s programming experience. They had previously done big projects in C, C++ and Java, whereas they didn’t really have much knowledge of the other programming languages. It was therefore decided that it would be best not to start learning something completely new, but choose one of the already familiar languages. As C++ is a newer, object-oriented version of C, the question was whether to choose C++ or Java. A brief comparison of the two languages is shown below.

<table>
<thead>
<tr>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>More or less backwards compatible with C source code.</td>
<td>Designed from the ground up without backward compatibility with any previous language.</td>
</tr>
<tr>
<td>Allows direct calls to native system libraries.</td>
<td>Call through the Java Native Interface.</td>
</tr>
<tr>
<td>Exposes low-level system facilities.</td>
<td>Runs in a protected virtual machine.</td>
</tr>
<tr>
<td>Optional automated bounds checking.</td>
<td>Always performs bounds checking.</td>
</tr>
<tr>
<td>Supports native unsigned arithmetic.</td>
<td>No native support for unsigned arithmetic.</td>
</tr>
<tr>
<td>No standardized limits or sizes for any numerical types. Only relative sizes specified.</td>
<td>Standardized limits and sizes of all primitive types.</td>
</tr>
<tr>
<td>Parameters passed by value, pointer or by reference.</td>
<td>Parameters always passed by value; however objects are accessed through references and it is those references that are passed or returned by value, not the objects themselves (comparable to using pointers for parameters in C++)[1]</td>
</tr>
<tr>
<td>Explicit memory management, though third party frameworks exist to provide garbage collection.</td>
<td>Automatic garbage collection only, though can be manually tuned by programmer.</td>
</tr>
<tr>
<td>Allows explicitly overriding types.</td>
<td>Rigid type safety except for widening conversions.</td>
</tr>
</tbody>
</table>

The C++ Standard Library has a much more limited scope and functionality than the Java standard library but includes: Language support, Diagnostics, General Utilities, Strings, Locales, Containers, Algorithms, Iterators, Numerics, Input/Output and Standard C Library. Users must choose from a plethora of (mostly mutually incompatible) third-party libraries for threads, network I/O, GUI, and more functionality than the barebone implementations provided by the C++ Standard Library.

Operator overloading

| Full, multiple inheritance | Full single inheritance, multiple inheritance from interfaces only |

Figure 3.4-1 C++ and Java comparison (Wikipedia 2008a).

What is most important in this comparison are aspects regarding memory and speed, as these were believed to be a very relevant issue in this project. Because Java provides so many extra functions compared to C++, such as garbage collecting, and even more importantly runs via a Java Virtual Machine, programs written in Java work slower than those written in C++. Another thing that favours the use of C++ over Java is explicit memory
management – thus memory can be organized more efficiently than in the automatic garbage collecting Java environment. It were generally these two issues that led to the choice of C++ as the project’s main programming language.

Once it was decided to write the image organisating application in C++, it was necessary to choose a library for creating the GUI. Several options were considered, with most thought given to the most popular – WinAPI, Motif and QT. The biggest drawback of the first one is no portability – an application developed with WinAPI will only be useful under Windows, and never anywhere else. The biggest drawback of the second library is very poor and scarce online documentation, which leads to the need of buying books about it and makes chances of finding online solutions to encountered problems scarce. Eventually, the QT library (v 4.4.2) was chosen, because of the following reasons (Nokia Corporation 2008, Nokia 2008a):

- It is free (Under the GNU General Public Licence)
- It is a portable library
- The toolkit comes with numerous examples and demos which make learning its use much easier
- There is extensive and very precise documentation available online
- There are numerous online forums discussing different problems encountered while programming with QT
- QT offers many extra features apart from easy GUI creation, most notably: thread support, network support, file support, openGL support and database support – these could turn out a great help in case any would be needed in future work, and any extra libraries wouldn’t need to be included

Eventually it can be said, that looking at the whole process of developing the application now that it is finished, the choice of QT seems very good.

The last thing that had to be decided as far as programming tools are concerned is the software development platform. There are a few possibilities when programming in C++. The ones that were considered are Microsoft’s Visual Studio, Dev-C++ and Eclipse. It probably would have been a difficult choice were it not for QT. As it is, QT is very difficult to compile with Visual Studio, and actually also with Dev-C++. With Eclipse however, QT compilation doesn’t pose the slightest problem as there is a QT plug-in written specially for this platform (Nokia 2008b). Therefore, all there is to do is install it.

This simplicity of integrating QT with C++ is the main reason for which the Eclipse platform was chosen as the preferred software development platform.
3.5. Development Methodology

Before discussing the application’s design, it is important to mention the set of procedures (methodology) that was chosen to achieve a satisfactory final version of the product.

It is believed that the general functionality of the application can be clearly agreed upon with the client during the first interview. All changes that would be implemented in the program later on could perhaps be even of big significance, but wouldn't affect the system's architecture as a whole. Therefore, the application’s architecture could be designed at the very first stage of the development process.

As to the methodology in general, the most fitting term describing it would be software prototyping. Although a great majority of the requirements could be collected during the first two interviews with the client, using a working prototype of the application is something completely different from theoretical discussions. It is essential to give the user a chance of expressing their opinion regarding different features of the application before the final version is released – thus the client can express their wishes regarding things they would like improved in the prototype, and so be more satisfied with the final outcome.

It was decided that the development of two prototype versions before releasing the final version of the application would suffice. Perhaps in architecture-complex network related projects it would have been better to create more prototype versions, but here it was deemed unnecessary. It was further believed that the client would demand any major changes only regarding the first prototype. The general sequence of developing the application is shown on the next page in figure 3.5-1.

After showing the client each of the prototype version of the product, they were given a few days to test it. After that the client provided feedback regarding the parts of the application that they were asked about. This feedback was later analysed and any required improvements or changes were implemented. Therefore, the 1st prototype was the foundation for the 2nd prototype, and the 2nd prototype was the foundation for the final version of the product.

Although prototyping requires some effort in terms of interaction with the client and creating consecutive versions of the product, it is the design and implementation stage that required most work on the developer's part. Creating the software so that it worked properly, fulfilling the general requirements provided by the client at the beginning, was the most difficult and time consuming part of the development process. Therefore, planning tasks in detail and keeping track of the schedule was of utmost importance, and a key factor to eventual success.
Once the methodology was decided about, it was time to design the architecture of the application. This task included both designing the general architecture of the system as a whole, and making key decisions regarding what and how would be implemented.
4. Design

4.1. Concept of Similarity Sorting

Before deciding about the program’s general architecture and functionality, it is necessary to consider certain characteristics of the application, which will make it capable of sorting images by similarity.

In order to sort images by similarity, two things are required. First of all, the image content has to be somehow described. The image itself doesn't provide any useful information about what is in it, apart from the values of all pixels. If it were possible to somehow transform all these pixel values into a more simple form – for example an vector of numbers, then sorting these arrays of numbers could be considered.

The second obvious necessity is a clever way of performing this sorting. As arrays of numbers are multidimensional data, simple algorithms such as quicksort intended for one dimensional data are useless. An algorithm capable of sorting multidimensional data, preferably in a two dimensional space (as the images are to be displayed on a 2-D surface – the computer screen) is required.

The two necessary components and their relation with similarity sorting is shown on the diagram below.

![Diagram of Similarity Sorting Concept](image-url)

Figure 4.1-1 Similarity sorting concept
4.2. Image Feature Description

Not so long ago, describing image features in a sensible way could have been considered a very difficult task. However, during the last decade a new standard for describing multimedia content has emerged. This standard is MPEG-7, formally named *Multimedia Content Description Interface* (MPEG 2008). It is widely regarded as an international standard for multimedia description.

Among many other things, the MPEG-7 standard provides an elaborate specification regarding image content descriptions. It defines objects called *descriptors*, which are meant to describe different image features. MPEG-7 specifies both how to calculate these descriptors (which are eventually nothing else than sets of numbers) and how to compare these descriptors with one another. Therefore, it seems a perfect solution for what is needed within this project.

The amount of descriptors that the MPEG-7 standard offers is extensive. The general categories of existing descriptors referring to images are (MPEG 2008):

- Colour descriptors
- Texture descriptors
- Shape descriptors

It has to be chosen which ones will be tested and implemented in Image Organizer. In order to make a good choice, all categories were analysed in terms of how they reflect the visual impression that the human eye gets when looking at an image. The conclusion was that the most useful group to consider are colour descriptors.

There are seven colour descriptors defined by MPEG-7 (for more information see Modi 2009, Szymczyk 2009). However, not all are suitable for accurate representation of the image content. Furthermore, not all can be easily compared with each other in terms of distance (which is necessary for the purpose of sorting). Having these criteria in mind, two descriptors were chosen to be implemented in Image Organizer as means of describing image features. These descriptors are:

- Color Layout descriptor
- Scalable Color Descriptor


4.2.1 Color Layout Descriptor

The Color Layout Descriptor (CLD) provides information about the spatial distribution of colours in the image. It stores information regarding which colours appear in the image and where they appear.

The extraction process of this descriptor involves several stages. It is shown in figure 4.2.1-1.
The first thing that is done with an input image is resolution scaling. The image is transformed into an 8x8 miniature. Each pixel in this miniature has the colour being the average of all initial pixels that it represents from the original image. This is shown on figure 4.2.1-2.

After this, pixels from the image miniature are transformed from the RGB color space into the YCbCr color space (this can later enable separating luminance and chrominance coefficients of the resulting descriptor). In the next step each YCbCr component is separately
transformed with the use of a DCT transform. This yields three 8x8 arrays of transformed YCbCr coefficients. The coefficients are quantized and afterwards placed into three linear arrays by zigzag scanning, which orders them as shown on the picture below:

Figure 4.2.1-3 Concept of zigzag scanning (Galiński 2008)

Zigzag scanning stems from the nature of the DCT transform – for more details see Wikipedia 2008b.

Once all coefficients are placed in three arrays in the desired order, the first few coefficients from each array are chosen to form the CLD descriptor. It is advised to take the first six Y coefficients and three of each Cb and Cr coefficients to form a descriptor made out of 12 coefficients (MPEG 2002).

The last thing that needs to be mentioned is how to calculate the distance between two descriptors. MPEG defined this as the sum of Euclidean distances between each Y, Cb and Cr component. If there are two descriptors CLD1 (YCoeff, CbCoeff, CrCoeff) and CLD2 (YCoeff', CbCoeff', CrCoeff'), then the distance between them will be:

\[
D = \sqrt{\sum_{i=0}^{\text{Max\{NumberOfYCoeff\}-1}} \lambda_{Y}(YCoeff[i] - YCoeff'[i])^2} + \sqrt{\sum_{i=0}^{\text{Max\{NumberOfCCoeff\}-1}} \lambda_{Cb}(CbCoeff[i] - CbCoeff'[i])^2} + \sqrt{\sum_{i=0}^{\text{Max\{NumberOfCCoeff\}-1}} \lambda_{Cr}(CrCoeff[i] - CrCoeff'[i])^2}
\]

Formula 4.2.1-1 Distance between two CLD descriptors (MPEG 2002)

The \(\lambda\) symbol is the weight of each coefficient taken into the calculations. MPEG suggests the following weight assignment:

<table>
<thead>
<tr>
<th>(X)</th>
<th>Coefficient Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>2 2 2 1 1 1</td>
</tr>
<tr>
<td>Cb</td>
<td>2 1 1</td>
</tr>
<tr>
<td>Cr</td>
<td>4 2 2</td>
</tr>
</tbody>
</table>

Table 4.2.1-1 Weighting values for the default descriptor (MPEG 2002)
4.2.2 Scalable Color Descriptor

The Scalable Color Descriptor (SCD) is a colour histogram providing information about the quantity of each color in the image. Unlike the CLD descriptor, SCD doesn't contain any information regarding where each of these colours is placed in the image. Extraction of the Scalable Color Descriptor can be depicted as a process consisting in several stages – similarly as in case of Color Layout. It is shown in figure 4.2.2-1.

- **hue** – is in the range of 0-359 and represents degrees on the color wheel as shown below.

![Figure 4.2.2-1 Scalable Color Descriptor extraction process](image-url)
Figure 4.2.2-2a HSV Color Wheel

- saturation – is in the range of 0-255 and represents how strong the colour is, the more gray the colour, the closer S is to 0

Figure 4.2.2-2b HSV saturation scale

- value – is in the range of 0-255 and represents the brightness of the colour, the darker the colour is the closer V is to 0

Figure 4.2.2-2c HSV value scale

For the purpose of calculating the Scalable Color Descriptor, the above HSV values are quantized, so that they can fit into an 256-bin structure, which can be illustrated as a cylinder (figure 4.2.2-3). Each dimension is quantized in the following way:

- H – hue to 16 values
- S – saturation to 4 values
- V – value to 4 values

Figure 4.2.2-3 The HSV color model (Galiński 2008)
After this quantization a 256-bin HSV histogram is created. Then the histogram values are in turn quantized. For more details regarding this see MPEG 2003.

The next step is a series of 1-D Haar transforms. These transforms generally add and later subtract two different coefficients from each other, and save the sum as the first coefficient, and the difference as the second coefficient. This process is repeated until there is only one sum-coefficient left. After that the resulting values (1 all-sum coefficient and 255 difference-coefficients) are saved in a 256-cell array in a special order which ensures, that coefficients containing more information are at the beginning. Eventually this yields a set of 16 low-pass coefficients (most important) and up to 240 high-pass coefficients.

All of these coefficients constitute the Scalable Color Descriptor. However, as the descriptor is scalable, it is up to the developer how many coefficients they need, and so the SCD descriptor can be represented by any number of values ranging from 16 to 256. Another thing that is often done for storage efficiency is discarding less significant bits of all coefficients. However, when using the descriptor in an image sorting application for the purpose of comparing images, this option will most probably not be made use of, as image description accuracy needs to be as high as possible.

As to calculating the distance between two Scalable Color Descriptors, this should be done in a different way than in case of the Color Layout Descriptor. It is recommended to use the L1 norm for this (MPEG 2002). This means that the distance between two descriptors is the sum of differences between pairs of corresponding coefficients. If we therefore have two descriptors SCD1 and SCD2

\[ D = \sum_{i=0}^{\min(\text{NoOfSCD1Coeff}, \text{NoOfSCD2Coeff})} |SCD1[i] - SCD2[i]| \]

Formula 4.2.2.1 Distance between two SCD descriptors

As two different SCD descriptors may have different numbers of coefficients, calculating the distance between them should be done considering only those coefficients that appear in both descriptors.

It was decided that descriptors describing images should be calculated at the very beginning of the sorting process – when only the image is loaded into the program. Only this way efficient sorting of the images and the associated descriptors may be possible.
4.3. Similarity Sorting Algorithm

Once it has been decided how to describe image features with arrays of numbers, the algorithm to be used for sorting these arrays should be considered. There are a number of algorithms that are capable of sorting multidimensional data. One whole group of algorithms fits into the category of multidimensional scaling (Boratti 1997, StatSoft 2008, Wikipedia 2008c). The FastMap algorithm used in the image sorting program Emir (analysed in section 3.2.2) also belongs to this category. In general Multidimensional Scaling is an interesting way of sorting data, but has one big drawback. This is the fact, that data is organised in a continuous space, be it one, two or 3-dimensional. Therefore, in case images which are needed to be displayed, it could and probably most of the time would happen, that many different images would be mapped so close to each other, that their display would be impossible.

A completely different approach to sorting multidimensional data is offered by the Self-Organising Map algorithm developed by Prof. Teuvo Kohonen in the early 1980s (Honkela 1998). This iterative technique is based on the creation of a map of nodes and such an arrangement of these nodes, that most similar ones appear next to each other. As such, this algorithm has one big advantage over multidimensional scaling – images can be matched to nodes and easily displayed on a grid of these nodes, without overlapping.

After a slightly more detailed analysis of the available multidimensional data sorting algorithms, it was decided that the Self-Organising Map (SOM) algorithm would be best because of the following reasons:

The SOM algorithm comes in two variations: the classic incremental version, and a new batch version intended for use on computers. Both versions of the SOM algorithm are presented and shortly explained below.

4.3.1 The Incremental Self-Organising Map Algorithm

The Incremental SOM algorithm uses a neural network to enable multidimensional data organization in any chosen number of dimensions. As for the purpose of image sorting a target 2-D sorting space is required, this is the case that the explanation will be focused on.

The SOM algorithm takes place in three stages.

1. Initialization of node map
2. Training of node map
3. Matching input data to trained node map

The first stage involves the creation of a map of nodes, each node being of the same data type as the input data that is intended to be sorted. If colours were sorted, then the data type would be a colour (3 dimensions – R,G,B), and the visualization of such a node map (in 2-D) could look as shown in figure 4.3.1-1.
In case of sorting images via descriptors, each node will have to be a descriptor itself. The number of nodes should be equal to at least the number of input elements that are wished to be sorted. The created initial node map can contain any values in each node, even random ones. However, better initialization of the nodes in the initial map will cause the algorithm to produce good results more quickly.

The next stage is training the node map with input data. This is a process performed in a series of iterations – typically 5-10 times as many iterations as there are input elements to be sorted. The training is performed according to the following algorithm (Germano 1999):

For $t$ from 0 to 1
   Randomly select a sample unit (input element)
   Find best matching unit (BMU) in node map
   Scale neighbours of the found BMU
   Increase $t$ a small amount
End for

Each iteration consists of several steps. The first is quite simple – just selecting a random element from the data that is wished to be sorted. The next thing that should be done is finding the best matching unit in the trained node map. This means finding the node, the distance of which from the input element is the smallest. Therefore, in case of the CLD descriptor, the Euclidean distance as in formula 4.2.1-1 would be calculated, whereas in case of the SCD descriptor formula 4.2.2-1 would be used.

The next step is determining the neighbours of the best matching unit. The SOM algorithm requires that the number of these neighbours decreases with time (as $t$ increases), so a common approach is starting with a fixed neighbourhood radius of $R$ and decreasing it monotonically to 0 as $t$ reaches 1.

The final step is training the nodes in the neighbourhood – making them more similar to the input element that has been selected for the current iteration. This training needs to be performed with different weights. First of all, nodes closer to the best matching unit should be made more similar to the input element than nodes that lie further away. There is a number of functions possible to use here, but the most popular is the Gaussian function. Secondly, the amount that all nodes may learn should decrease with time (as $t$ increases). This decrease is usually implemented as linear. These two factors contribute to produce a weight $W$, which is used for training each BMU’s neighbour in the training process. An exemplary

---

Figure 4.3.1-1 Example of SOM node map (Germano 1999)
formula demonstrating how to calculate the value of W with the use of a Gaussian neighbourhood weighting function is shown below (Kohonen 2005):

\[ W(r_i) = \alpha(t)e^{-\frac{|r_0 - r_i|^2}{2\sigma^2(t)}} \]  

Formula 4.3.1-1 Calculating learning weights

where

- \(0 < \alpha(t) < 1\) is the learning-rate factor decreasing with time
- \(r_i \in \mathbb{R}^2\) is the vectorial location of the trained node in the display grid
- \(r_0 \in \mathbb{R}^2\) is the vectorial location of the BMU in the display grid
- \(0 < \sigma(t) < R\) is the width of the neighbourhood function, decreasing with time

After W is calculated, the neighbour node (among neighbours also the BMU itself) is updated according to the following equation (Formula 4.4.1-2):

\[
\text{NewNodeValue} = (1 - W) \times \text{OldNodeValue} + W \times \text{InputElementValue}
\]

After a certain number of iterations a map containing values similar to those of the input data will form, and it will have the trait that nodes close to each other in any given region will be very similar.

The last thing that needs to be done to get the input data sorted, is matching the input data to the nodes of the trained map. If the training had been performed well enough, it should be possible to associate each input element to a well-matching node on the trained node map. After this, the input elements are placed on a grid that corresponds to a node map, and so are sorted in the way it was desired.

For more information see Kohonen 2005a, 2005b, Douglas 1999 or FAQS.ORG 2008.

4.3.2 The Batch Self-Organising Map Algorithm

The main drawback of the incremental-learning SOM algorithm is the fact that it is slow with large data sets. Hence, a new version of the SOM algorithm, known as the Batch-SOM, has been developed. It is believed to work faster on computers, especially with MATLAB functions. Of the three main stages that are necessary to obtain input data in a sorted fashion, the first and last are the same as in case of the classic SOM algorithm. The
difference lies in the training of the node map. In case of the Batch-SOM the training is done in the following steps (Kohonen 2005a):

1. For each node $i$ on the map, collect a list of all those input elements, whose BMU is in the neighbourhood of node $i$
2. For each node take the weighted mean over the respective list
3. Repeat step 1 and 2 several times

In this batch version, $t$ is not incremented each time one BMU is found and it’s neighbourhood trained, but it is incremented after all BMU’s neighbourhoods are updated once. The other difference is how each node is updated. It is not done as in case of the incremental SOM algorithm – consecutively with the use of formula 4.3.1-2. Instead, each node is updated once, with all affecting BMU’s near it, according to the equation below (Kohonen 2005a):

$$m_i(t+1) = \frac{\sum_{j=1}^{n} W_{ic(j)}(t) \cdot x_j}{\sum_{j=1}^{n} W_{ic(j)}(t)}$$

Formula 4.3.2-1 Batch SOM training

where:
- $n$ is the number of input data elements
- $m_i(t+1)$ is the new value of node $i$
- $c(j)$ is the BMU of sample vector $x_j$ (from input data)
- $W_{ic(j)}$ is the weighting factor of the neighbourhood function

Both the incremental-learning SOM algorithm and the Batch-SOM algorithm have decided to be implemented and compared in order to choose the best solution for Image Organizer. More details regarding the implementation will be provided in chapter 5.
4.4. Keyword Filtering

4.4.1. Functionality

A considerable amount of thought has been given to the issue of making image sorting and retrieval easier with the help of keywords. A number of possibilities were considered, but eventually it was decided that the most sensible approach would be providing keywords as a mean of filtering. In other words, the user could specify what conditions they wish the desired images to match and load only those images that comply with it.

Another option that was deemed useful was the possibility of filtering images already loaded and displayed in the program, perhaps even already sorted. For instance sorting 5000 images by similarity and then leaving only those that correspond to, for instance, mountains should make it really easy for the user to find what they are looking for.

A different matter is how keyword matching should actually be performed. Should matching part of a keyword also count as a match? Inevitably every solution in this matter has its drawbacks. It was eventually decided that a “match” is recognized if the keywords saved for an image contain the user-specified keyword. This means that if a keyword describing an image is longer, for example “Long Island”, then the word “Island” will also count as a match. Nonetheless, entering free-text keywords alone could be tedious and would force the user to remember all that they had entered previously, possibly during several years. To avoid this obvious disadvantage, structured categories have been decided to be introduced apart from free-text keywords. For best description efficiency the categories should be hierarchical – so categories should have subcategories, and those subcategories could also have further subcategories. This way the user can choose to describe an image as for example nature->landscape->mountains, which provides means of filtering at different levels. An exemplary diagram of how the category and free text descriptions are structured is shown below.

![Image: Diagram showing category and keyword structure]

Figure 4.4.1 Image category and keyword description
4.4.2. Storing technique

Apart from issues regarding the functionality of keywords and categories, it has to be decided how the keywords and categories describing images will be stored. There are generally two possibilities:

1. Storing them separately from the images as a data file or a real database
2. Storing them within the image files

The second possibility stems from the fact that most image formats include specific blocks intended for storing application-specific data or text comments. These places in the image files can be therefore used by Image Organizer for storing keywords and categories.

Obviously both possibilities of storing keywords have advantages and disadvantages. Eventually, after a detailed analysis of the problem, it was decided that the second possibility – storing keywords within the image files themselves – will be implemented. This is generally because the following advantages of this approach:

- The data is safe within the image file and doesn't disappear when the image is moved, renamed etc.
- The data can be accessed at any time when only the image is available, there is no need to create an additional database, so the structure of Image Organizer remains more simple and consumes less space
- Extraction of the data when loading images is quick, as the whole database doesn't need to be searched each time a specified image is loaded from the hard drive

The exact technique of storing keywords and categories will be described in slightly more detail in the implementation chapter.
4.5. General program architecture

Once the important aspects regarding advanced functions of the program are settled, all components can be put together into one whole, and the architecture of the whole system specified. From the broadest point of view, Image Organizer is intended to be an application capable of transforming input data in form of images into a somehow organised sequence. The input data should be first chosen by the user from a digital library. A diagram of the whole procedure is shown below.

![Image Organizer general architecture diagram]

Figure 4.5-1 Image Organizer general architecture

The above diagram gives just a vague overview of what the program is capable of doing. In order to start writing the program itself, it is first necessary to specify more precisely which functions will be available at which stages, and what the user will have available depending on the state in which the program is. Therefore, a more detailed diagram of the application’s functionality has been prepared.
As it is visible on the activity diagram, the program can generally be in one of two different states: without any images loaded into the program or with some images loaded into the program. The latter can further be divided into displaying images in the order that they were loaded.
loaded, or in an already organized fashion. A quick overview of the functions available to the user in each of these stages is presented below.

1. State when there are no images loaded into the program

This is the initial state after the application has been started. No images are displayed, and it is necessary to load some into the program. This can be done by selecting a folder or single image through a special directory view visible on the main panel of the application. It is possible to filter the loaded images according to keywords or categories that describe them, and load only images that match a certain specification, or exclude chosen images from being loaded.

Another thing that is available to the user at this stage is changing the program options. These options will include things such as whether or not to load images from subdirectories of chosen folders or whether or not to enable keyword filtering and editing in the program (disabling it benefits the program’s speed).

2. State when there are some images loaded into the program

Only once there are some images loaded into the program, it becomes fully useful. There is a number of things that the user may try at this point.

First of all, it is possible to sort images. Whether the displayed images are sorted or not, it is possible to select a subset of images. After doing so, it is possible to do different things with the selected images, such as copying them to another location, deleting them or editing keywords that describe those images.

Apart from doings things with selected images, it is possible to filter all the images that have been loaded into the program according to keywords, load more images into the program, or clear those images that are already loaded. In the last case the program will return to the first state – when no images are being displayed.

The last not yet mentioned possibility is changing program features, as described in point 1. It is something that can be done at any point when using the application.
4.6. Main Panel GUI

One more thing that inevitably had to be decided about before implementation could be started is the layout of the main panel in the application – the panel that appears after the application is started. It should be designed to make image loading, sorting and viewing as easy as possible. To achieve this, some solutions from software described in section 3.2 have been used in this design.

It was decided that the image displaying panel should be placed in the centre of the screen and take up most space – it is images that are most important so they should be shown in as much detail as possible. A tree-like view of directories allowing to add more images could be placed on the left hand side, along with a space underneath it to display miniatures of selected images. The keyword panel could be placed on the right, while a menu bar allowing the selection of a sorting way quickly could be placed directly under the main menu bar. The whole prototype of the main panel was designed as shown below.

![Main panel GUI prototype design](image-url)
5. Implementation & Testing

The implementation process is the culmination point in the project, when work completed as part of earlier stages is proved either useful or worthless. Implementation involves gathering all information obtained from the client together with outcomes of the research and design stages, and putting it all together in order to create the product that was desired. Therefore, implementation requires good focus on currently completed tasks in order to make work efficient.

Something that often arises at the implementation stage, especially in case of software creation, are unforeseen problems and difficulties. These usually need to be solved in the course of work, as soon as they arise. However, in order to notice these problems, frequent testing of the currently developed prototype is necessary. Usually the earlier a problem becomes known, the easier it is to solve it. Therefore, the application developed within this project has been tested very often, virtually every time a new feature was implemented. This allowed to notice both actual and potential problems very quickly.

All problems that have been revealed during the implementation process of this project will be mentioned and the found solutions explained later on in this chapter.

The hardware used for the implementation and testing was a Intel Core 2 Duo 2,4 GHz laptop with 3 GB of RAM and virtually unlimited space on the hard drive (more than could be required).

5.1. Program Structure

The first created element was the GUI – all windows, panels, buttons, menu options etc. To complete this task properly it was necessary to create a list of all main classes that would appear in the program and their functions. This way work wouldn’t be wasted because of the need to shift parts of code among different files. Of course, sometimes this has to be done, but with better planning more time is saved. The list of all key classes constituting Image Organizer (in the first approach) is presented below.

<table>
<thead>
<tr>
<th>class name</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColLayoutDescriptor</td>
<td>Stores Color Layout Descriptor (CLD)</td>
</tr>
<tr>
<td>ColLayoutExtractionUtil</td>
<td>Extracts CLD calculates distance between CLDs</td>
</tr>
<tr>
<td>ImageItem</td>
<td>Storing ImageItem – the loaded image along with descriptors and other data describing it</td>
</tr>
<tr>
<td>ImageScene</td>
<td>Defines the structure responsible for displaying images</td>
</tr>
<tr>
<td>ImageView</td>
<td>Defines the structure responsible for viewing the ImageScene (QT specific requirement)</td>
</tr>
<tr>
<td>KeywordDialog</td>
<td>When choosing to edit keywords of selected images, a new dialog will appear to enable this</td>
</tr>
<tr>
<td>KeywordUtil</td>
<td>Performs operations on image files manipulating keywords</td>
</tr>
<tr>
<td>LoadImagesUtil</td>
<td>Loads images once the user chooses to do that</td>
</tr>
<tr>
<td>Main</td>
<td>Initializes the application</td>
</tr>
<tr>
<td>MainWindow</td>
<td>Most important class, displays the main window of the application, connects all classes together</td>
</tr>
<tr>
<td>ResizeImagesUtil</td>
<td>Reloads images in higher resolution when zooming is performed</td>
</tr>
<tr>
<td>ScalableColDescriptor</td>
<td>Stores Scalable Color Descriptor (SCD)</td>
</tr>
</tbody>
</table>
To get a better idea of the actual structure that the developed application possesses, a class diagram showing all the classes and their mutual relations has been created. It is presented below.

![Image Organizer simplified class diagram](image_url)

Figure 5.1-1 Image Organizer simplified class diagram

For the purpose of readability only inclusion in form of arrows has been shown on the above diagram. Apart from inclusions, many classes simply have access to other classes. For
example, the image sorting utility (SortImagesUtil) must have access to ImageItems as well as both descriptor extraction utilities, and descriptors storing classes, otherwise sorting couldn’t be performed. For more information regarding the details of class structure and relations see the source code included on the accompanying CD.

5.2. GUI Overview

The GUI created for the first prototype was generally very close to what was present in the final version of the product. The main panel looked as designed in section 4.6. An example with some images loaded into the program looks as follows.

![Figure 5.2-1 First prototype - main panel view](image)

It matches the layout that was designed quite closely. There is a directory view on the left side of the main window, which allows loading chosen images by either double-clicking on them or clicking on the folder while holding the SHIFT key. A miniature of the currently selected image is displayed in the bottom-left corner of the screen, and the keyword panel is on the right. It is possible to select several of the displayed images by clicking on them while holding the SHIFT key and then it is possible to do several things with them via a context menu that appears when the right mouse key is pressed. This menu looks as below.
All options apart from 'Select all images' are unavailable, as no images are selected. Once some images get selected, all options in the context menu become available.

If that were the case, and user chose 'Move selected images', a dialog box asking to specify the location would appear in the middle of the screen, as presented below.

One more interesting thing regarding the GUI is the dialog that appears when editing keywords of the selected images is chosen. Similarly as in case of moving images, the dialog appears in the middle of the screen and looks as shown in figure 5.2-4.
The keyword editing dialog enables custom editing of keywords and categories for a single image or for all the selected images at once.

Another thing that needs mentioning is how the user can move the displayed images. It is possible to drag all images in a chosen direction. This will scroll them in such a way, that images moving out of sight at the top will reappear at the bottom and vice versa. The same applies to left-right movement. All this allows to place a chosen region of all displayed images in the centre of the screen, which is important in case of similarity sorting. A illustration of this possibility is shown below:
Apart from scrolling, it is possible to quickly magnify a chosen region of the displayed images by zooming with the mouse wheel or the buttons available on the left side of the top menu.

5.3. Main functionality implementation

A lot of the basic functionality in the first prototype of Image Organizer is simply rudimentary programming work, which has to be done with the use of the Qt library. All the classes shown in figure 5.1-1 need to be created with the necessary variables and functions. It is beyond the scope of this report to go into the details of this, but a number of things do need to be mentioned. These include the implementation of image descriptors, the implementation of the similarity sorting algorithms and the specifics of saving keywords into image files.

5.3.1. Descriptors

Both the Color Layout Descriptor and the Scalable Color Descriptor have been implemented as the MPEG-7 standard implies. The extraction processes of both descriptors have been described in the design chapter. This step-by-step extraction description was the foundation for actually writing the code to calculate descriptor values.

It is also important to note, that a very helpful tool making the implementation a lot easier is the MPEG-7 XM Reference Software (MPEG 2003). It is a big library written in C++ capable of calculating all MPEG-7 defined descriptors. Its source code was a big help, although it was not copied as a whole. All the code in Image Organizer is re-written and only based on that from the XM model. This allows to make use of Qt-specific templates, and to suit the image organising application's needs better.

Image Organizer possesses four separate classes to handle descriptors. How all the interactions take place in the application is shown on the following diagram.
5.3.2. SOM algorithm

Both the incremental learning SOM algorithm and the batch SOM algorithm have been implemented to follow the procedures described in the design chapter in section 4.3. All the code was written from scratch, although the examples in Kohonen 2005a, 2005b and Germano 1999 were found very useful to speed up the implementation process. Throughout the implementation of the SOM algorithms Qt templates such as a variable-length array template QVarLengthArray were made use of. This is because Qt templates and containers are guaranteed to work at least as fast as STL tools and offer much more functionality.

5.3.2.1. Incremental Learning SOM

The Incremental SOM algorithm was implemented so that it complies the model described in literature, created by Prof. Kohonen. The main body of the algorithm based on the CLD descriptor (code for SCD is similar) is shown below – as it is written in Image Organizer:

Figure 5.3.1-1 First prototype – diagram of classes using image descriptors
int size, rows, columns;
size = (*imageItemVector).size();

columns = rows = (int) (ceil(sqrt(size) + EXTRA_ROW_COL));
createCLDNodeMap(rows, columns);

switch (INIT_MODE) {
    case 0: generateRandomValuesForCLDNodeMap(rows, columns);
        break;
    case 1: generate4ValuesForCLDNodeMap(rows, columns, false);
        break;
    default: generateRandomValuesForCLDNodeMap(rows, columns);
}

int currentSample = 0;
do double T_INC = 1.0f / NUMBER_OF_ITERATIONS;
do double t = 0.0;
/*...*/
while (t < 1.0) {
    //Get consecutive image descriptors from the vector
    CLDDescriptor *rSample =
        (*imageItemVector)[(int) currentSample % size] ->getCLD();

    //Find its best matching unit
    QPoint bmuLoc = getCLDBmu(rows, columns, rSample);

    //Scale the neighbours according to t
    scaleCLDNeighbours(rows, columns, &bmuLoc, rSample, t);

    //Increase t to decrease the number of neighbours and
    //the amount each weight can learn
    currentSample++;
t += T_INC;
/*...*/
}
/*...*/
matchImagesToCLDSOM(rows, columns);

Figure 5.3.2.1-1 Source code fragment – Incremental SOM body

The parts of code which are not directly related the functionality of the SOM algorithm have
marked as /*...*/ for better clarity.

The first line of code just initializes variables. These variables have values assigned to them
in the next two lines. *imageItemVector* is a pointer to a vector containing all the images
loaded into the program along with data describing those images, so also descriptors. All
those images will eventually be displayed on a 2-D surface of nodes – so a grid with rows
and columns. The number of rows and columns needs to be chosen so that all images can be fitted into this grid without overlapping, and there need to be some extra spaces to ensure that the SOM algorithm works properly. This is why when rows and columns are initialized, the \textit{EXTRA\_ROW\_COL} parameter is used.

The next lines of code perform other initializations. The \textit{createCLDNodeMap()} function creates the structure which will be updated (trained) during the SOM algorithm. This structure is a simple 2-D array (with the number of specified rows and columns) containing pointers to descriptors. This array of descriptors needs to be initialized before the SOM algorithm starts training – it is performed in the \textit{switch} block. As mentioned earlier, this initialization may be random, but it is better when it is not. A good initialization is placing four completely different descriptor values in four corners of the trained node map and calculating a weighted average for all those in between – this is done in Image Organizer when \textit{INIT\_MODE} equals 1.

The next initializations include the step by which the time parameter \(t\) will increase. As \(t\) changes from 0 to 1, this step has be \(1/n\) divided by the number of iterations.

Once all initializations are finished, the training is started. It proceeds as described in the design chapter in section 4.3.1. During one iteration the following steps are completed:

- A CLD descriptor from the image-item vector is chosen
- Its best matching unit (node) is found on the CLD node map
- Neighbours of this BMU are scaled to be more like the input descriptor
- The time factor \(t\) is increased

The last stage shown in the above source code fragment is matching input data to the trained node map. This is done in a quite sophisticated way in order to ensure that all images are matched well. Every image is associated with ten nearest (in terms of descriptor distance) nodes. After this, all images are matched to single nodes on the trained map in such a way, that the sum of all distances between matched images' descriptors and the nodes to which they are matched is smallest. This avoids the situation when certain images are matched to nodes very well and others are matched completely wrong. Once all the matching is finished, the final result may be displayed.

\subsection*{5.3.2.2. Batch SOM}

The Batch SOM algorithm was implemented based on its descriptions in Kohonen 2005a and Kohonen 2005b. As mentioned in the design chapter, the difference between the incremental and batch SOM algorithm lies in the training stage. Therefore, only this stage will now be analysed. It is shown below (as previously, based on the CLD descriptor):
Here the time parameter $t$ is not increased with each consecutive iteration, but each time all nodes in the map have been updated once. One such ‘series’ of iterations therefore consists of:

- creating a map which will store lists (for each node on the map) of all those input data elements, the BMU of which lies the neighbourhood of the given node
- finding those input data elements and placing them in the created structure
- scaling each node on the map according to the created lists (done with the for block)
- updating the time parameter for the next series of iterations

For more precise information regarding the performance of each Incremental SOM or Batch SOM algorithm function, see the source code of the file somutilthread.cpp present on the attached CD.

### 5.3.2.3. Parameter choice

As has been previously mentioned, the SOM algorithm involves numerous parameters which can be chosen in many ways. In order to find out which configurations of these parameters are best for sorting images, it was necessary to perform many tests. To make changing parameters easier and avoid the need to re-compile the whole application each time a parameter needs to be changed, a ‘SOM parameter’ panel was introduced into the first prototype. It was there only for the purpose of testing and was removed in the second prototype. It was placed on the right-hand side of the main window, as an optional tab-widget, and looked as shown below.
The results of all SOM-related testing will be described in section 5.5.

5.3.3. **Keyword saving into image files**

As already mentioned in the design chapter, different image formats define special places in the image file where metadata can be stored. This possibility was explored in detail in order to find out exactly where data could be inserted depending on the image file format. An overview of the results is presented in the document “Keyword_storage” included on the attached CD. For more information you can also see Code Projects 2001, Augustyniak 2007, W3C 2008a, W3C 2008b, Portable Network Graphics 2009a and 2009b, Digital Camera Software 2009 and Maas 2003.
5.4. Encountered Problems

A number of problems arose in the course of implementation. These problems were not considered earlier, as they could not have been foreseen without actually trying how the implemented application works. The problems, along with any found solutions are shortly described below.

5.4.1. Program memory

Compression techniques ensure that images do not take up too much valuable space on our hard drives. However, when loading images into RAM memory, they are in an uncompressed form. If an image has 32 bits per pixel, then with a resolution of 3000x2000 pixels it takes up

\[3000 \times 2000 \times 32 \text{bits} = 92000000 \text{bits} = 24000000 \text{bytes} \approx 22.9 \text{MB}\]

It becomes clear that in case of loading images in full resolution, any computer will quickly run out of memory. One thousand photographic images already requires over 2 GB of RAM.

To solve this problem, it was decided to load into memory not the real images, but their miniatures. This way a lot of space can be saved. As a default Image Organizer loads miniatures in a resolution of 128x128, which means that with 2 GB of RAM available, assuming 32bpp in each image, over 32000 images can be loaded before the program runs out of memory.

Unfortunately loading miniatures of images into program memory makes zooming more complicated – it is necessary to reload the zoomed images to make them visible in a higher resolution. But it cannot be avoided as loading miniatures at the beginning is absolutely essential.

5.4.2. Loading speed

The second issue that was a big concern during the first stages of implementation is the speed of loading images. Loading 1316 low-resolution images of the JPEG format first took an unbelievable 17.3 seconds on very good hardware! This was totally unacceptable.

Fortunately introducing the concept of loading miniatures instead of full-resolution images greatly improved this matter. JPEG offers the biggest potential improvement as it codes images in scalable quality and scalable resolution, so when loading a miniature much of the image file doesn’t need to be read at all. Loading miniatures increased the loading speed more than three times. Nonetheless it still wasn’t impressive. Therefore, to further improve the loading speed, a special feature of the Qt JPEG decoder was used – scalable quality. Setting the quality parameter to the lowest value when loading image miniatures gave a further reduction of the loading time by half. Of course, once the images are zoomed in they can be reloaded in higher quality.

The results of all tests described above are shown in the graph below.
Figure 5.4.2-1 Image loading time comparison. Miniatures tested here were 64x64 pixels. In the final version of the product it was decided to load 128x128 miniatures.

Another thing that was decided to be useful is performing the image loading process in a separate thread. This way, the application doesn’t freeze when the loading is being performed and the user doesn’t have the feeling that something went wrong. Reloading images when zooming is also performed in a separate thread to ensure that zooming works smoothly. This is important especially in case of dealing with many high-resolution images.

5.4.3. Sorting speed & accuracy

The last problem, but the most serious one is the similarity sorting speed. The SOM algorithm turns out to be computationally very expensive. The algorithm is an iterative process, where the number of iterations should be generally specified as part of the input data. The more iterations are performed, the better the quality of the finally trained map will be. However, each iteration takes time, so from the time-efficiency point of view the number of iterations should be kept to a minimum. This means that accuracy and sorting speed are two things headed in different directions (figure 5.4.3-1). Therefore, a golden mean which will combine good sorting accuracy with reasonable sorting speed needs to be found.

Figure 5.4.3-1 SOM algorithm speed and accuracy

SOM Algorithm
There are a number of factors that may potentially improve the sorting speed without affecting the accuracy too much. These are as follows:

- Reducing descriptor size. As the SOM algorithm deals with arrays of numbers, each array being a descriptor, the less numbers the descriptor includes, the quicker all operations on it can be performed.
- Using the Batch-SOM algorithm. More regarding this is mentioned in the next section – Testing.
- Initializing the trained SOM map with values closer to the final result – this allows SOM training to focus on smaller areas at the very beginning, which shortens sorting time.

Reducing the descriptor size, as well as initializing the trained SOM map with better values gave significantly better results. More information regarding this is provided in the next section about testing.

5.5. Testing

All performed testing can generally be grouped into two categories.

1. Testing in order to find errors and ensure the implemented functionality works as it is supposed to.
2. Testing of different algorithms and solutions to find the best one.

The first type of testing was performed very frequently, virtually every time a new block of code was written. Revealed errors were very abundant, but that is normal in case of any programming-related task. All these minor errors are unimportant from the broader point of view and although fixing them was often difficult and required a lot of time, they will not be listed or mentioned in any more details.

The more important part of testing is related to the sorting efficiency and keyword support efficiency. As the latter is more straightforward, it will be described first.

5.5.1. Metadata saving and reading speeds

The specifics of where to save data in image files has been described earlier. But it is important to consider the time it takes to write the data into the image file, as also the time it takes to read it from the file.

When reading, the whole file actually needs to be processed in order to first find the place where the metadata was saved and then read it. Therefore, first all data in a file is read into the program and afterwards converted into a hexadecimal format. This makes finding the appropriate markers easy, and metadata can be read reasonably quickly.

Reading keywords from 1000 small resolution JPEG files (384x256) takes up less than one second. In case of high-resolution camera photos (4 Mega Pixels) this value reaches around 20 seconds, but is absolutely irrelevant compared to the image loading speed as a whole.

Saving metadata into image file poses more of a problem. This is because inserting anything into a file can only be done by re-writing the whole file from the point that new data was inserted. Nothing can be done to change this. The only time that can be saved is by re-
writing the file from the place that it absolutely has to be re-written, and not from the very beginning. This however is exactly what has been implemented in Image Organizer from the very beginning.

The resulting speeds of saving data in image files are not very good, but acceptable. Saving keywords in 1000 384x256 JPEG image files takes around 5 seconds. Saving them in 1000 4 Mega Pixel photos takes much longer – about 100 seconds. However, saving metadata is generally believed to be something the user does once for each image, so these times can be accepted. All the measured durations of metadata manipulation in image files are presented below.

![Metadata reading/saving time [s] for 100 images on Intel Core 2 Duo 2,4 GHz](image)

**Figure 5.5.1-1 Metadata manipulation durations**

### 5.5.2. Descriptors

Both the Color Layout Descriptor and the Scalable Color Descriptor have been compared in depth regarding the extraction speed, sorting speed and sorting accuracy that they provide. Results decidedly favour the Color Layout Descriptor.

To begin with, the extraction speed was compared. When loading 100 high resolution images the calculation of the CLD descriptor was absolutely unnoticed in terms of duration – so took less than 0.2 seconds. The SCD extraction however took considerably longer, amounting to about 1.5 seconds in total. This is not very important, but favours the CLD descriptor.

As to sorting speed, it depends on the amount of coefficients that are involved. The CLD descriptor could be reduced to just 9 coefficients – 3 of Y, 3 of Cb and 3 of Cr – without any visible deterioration of sorting quality. The SCD descriptor cannot be reduced beyond 16 coefficients (to make the information stored by it meaningful), so it consists of nearly twice as many coefficients as CLD. This means that there are twice more numbers to operate with, so the sorting time will be approximately two times longer.

The most important factor however is the comparison of sorting quality with the use of each descriptor. The following pictures show how each descriptor contributed to sorting a set of 1316 test images, each time using exactly the same SOM algorithm.
The conclusion is that Color Layout Descriptor yields much better results. Therefore, bearing all time-related and sorting quality related statistics, it has been decided to rely solely on the CLD descriptor in Image Organizer, not using SCD at all.
5.5.3. Sorting algorithm

The standard incremental SOM was compared with the Batch SOM in a series of tests. As the two algorithms operate in slightly different ways, having in common only the starting points and desired end results, it is impossible to compare specific parameters. The results have to be compared as a whole. In order to gain any useful conclusions it was decided that it would be tested how long it took each algorithm to achieve good sorting results with a test set of 1316 images.

The SOM algorithm produced a good result beginning from around 6000 iterations, which lasted around 2.5 seconds on the used hardware. This result is shown below.

The Batch-SOM produces satisfactory results slightly faster, but to have it produce results as good as those seen above it takes considerably longer, around 6 seconds for the same image set.
Perhaps the Batch-SOM would be faster with MATLAB functions (as is stated for instance in Kohonen 2005b), but implementing part of Image Organizer in MATLAB was impossible because of time constraints.

Further testing proved that it is possible to figuratively ‘combine’ the batch and incremental SOM algorithm. To be more precise, the incremental algorithm performs a certain number of iterations, increasing the time parameter \( t \) each time by small value until \( t \) reaches 1 (see section 4.3). Different, non-linear ways of increasing the value of \( t \) have been tried in search of more time-efficient solutions. And it turned out that increasing \( t \) by a step of around 0.15 two or three times at different training stages does not negatively influence the final sorting quality. This means that the number of iterations actually performed can be reduced by nearly 50%. As this was the most time-efficient solution found during the testing process, it has been eventually used in the first prototype of Image Organizer.

5.6. Feedback Regarding Prototype 1

After the first prototype version was finally ready, it was shown to the client and feedback was requested. Several days after this the client sent feedback by email.

5.6.1. Feedback analysis

The feedback is attached as Appendix B-2. The general conclusions regarding the requested improvements by the client are listed below:

- Order directories and files in the displayed tree-view better – according to type first and case-insensitive when sorting by name
- Provide the option of changing the background colour
- Start the Program Window as maximized
- Add a panel displaying information about an image that the mouse hovers above
- Show a miniature of the image that the mouse hovers above
- Enable selecting multiple images more easily
- Add the options of opening images and copying them to another location
- Add the options of hiding images and resorting those left
- Add options of sorting by name, date created and image size
- Add progress dialogs for time-consuming operations
- Allow adding new single images into a sorted grid of images without the need of resorting everything each time

5.6.2. Prototype 2 creation

All of these requests were analysed and fulfilled as best possible. This process will shortly be described in the rest of this section.

Improving the order of displayed directories was a simple task, only requiring adjusting some parameters of the Qt directory model. The achieved change is shown on the following picture.
To enable changing the background colour it was necessary to create a color choosing dialog that would appear when the appropriate option was chosen from the menu. The dialog offers a wide range of colours that can satisfy even topmost needs.
Starting the main program window as maximized is fairly straightforward and only requires setting this parameter in the MainWindow class. A bit more work was needed to provide the extra panel displaying information about images. It was deemed best to place this panel at the very bottom of the screen, and apart from information about an image that the mouse hovers above, add tips for the user – as shown in figure 5.6.2-3.

Whenever the mouse hovers above one of the displayed images, not only is information about the image displayed on the information panel, but a miniature of the image is also displayed on the miniature panel (as visible above).

The next request the client had was to enable easier selection of multiple images. To make this possible, it was decided that it should be allowed to stretch a rectangle by dragging the mouse when holding the SHIFT key, and make all images underneath this rectangle selected when the mouse is released. This way many images could be selected much faster than by single-clicking. The idea is demonstrated in figure 5.6.2-4.
As for the possibility of copying images to another location and opening images, both of these possibilities were added to the context menu showing when some images are selected and the right mouse key is pressed. Similarly, the options of hiding and resorting the images left were added to this menu:

![New context menu]

Figure 5.6.2-5 New context menu

Choosing ‘open’ causes the program to open all selected images in the default graphics editing program – in case Windows this is Paint. Copying takes place similarly to previously mentioned moving, though this time the old images are not deleted. Choosing ‘hide’ makes the selected images disappear and if resort is chosen after that, the remaining images will be sorted once again:

![Hiding images and resorting those left]

Figure 5.6.2-6 Hiding images and resorting those left
The client also wanted to have the option of sorting images more conventionally – by name, size and date taken. The buttons to do this were added to the top menu bar, next to the option of similarity sorting.

All of these sorting options were designed to make use of the fastest available algorithm – quicksort. This algorithm sorts even thousands of images in no time.

The progress dialogs that the client wanted were added to loading and sorting operations. This way the user of the application would always know when a time-consuming operation would finish. An example of such a progress dialog is shown below when images are loaded. A similar dialog appears when similarity sorting is performed.

The last thing that the client requested was the possibility to add single images into a grid of already similarity-sorted images, without resorting all images each time. To enable this the final form of the trained map at the end of the SOM algorithm was saved, so that it could be used for matching new images in the empty spaces. However, once the number of empty
spaces falls below a certain level (not guaranteeing a good match for new images), all images are resorted automatically.

5.7. Feedback Regarding Prototype 2

After the second prototype version was ready, it was shown to the client and feedback was requested, just like in case of prototype 1. Several days after this the client sent feedback by email. It was not as extensive as previously, but still important.

5.7.1. Feedback analysis

The feedback is attached as Appendix B-3. The general conclusions regarding requested improvements by the client are listed below:

- Hide unnecessary keyword-filtering panel not to confuse the user
- Make information about an image that the mouse is hovering above vanish as soon as the mouse moves away from it.
- Include the option of choosing whether or not to load images from subfolders of the selected directory
- Do not delete hidden images but keep them in program memory and enable showing them once again quickly
- Add the option of cancelling time-consuming operations
- Improve sorting speed for large image sets, and possibly image loading speed

5.7.2. Final Product creation

As in case of the previously obtained feedback, all the client’s requests were analysed and fulfilled as best possible in order to create the desired final product. This process will be described in the rest of this section.

The client’s first request was making the keyword filtering panel more clear and to hide all the filtering options when the user doesn’t need them. To meet this requirement, the keyword panel was made dynamic, so that it shows when the user selects keyword filtering, but is hidden otherwise. The improvement is shown below.
Figure 5.7.2-1 Improved keyword panel – hidden functionality when not required

The panel on the left shows the original design. The middle panel is the solution used in the final product – all keyword filtering tools are hidden when they are not needed. When however the user chooses to filter images that match a given category-keyword combination, the panel enabling this will appear. This is shown on the right.

The next requested improvement – making a miniature of the image and information about it vanish as soon as the mouse moves away from the image – was fairly simple to implement. It just required checking not only whether the mouse cursor entered the area where a new image is displayed, but also checking whether or not the cursor left this area.

In order to let the user choose whether or not to load images from subdirectories of the chosen directory, a checkable menu item was added to the Options menu.

Figure 5.7.2-2 New items in option menu
As it can be seen, the option whether or not to load subdirectories is not the only novelty to be implemented in the final product. Apart from that the user may choose whether they wish to enable keywords and two-stage sorting.

Disabling keywords causes keyword filtering not to be available for the user. The advantage of this is that metadata doesn't need to be read from image files when the images are being loaded – so the loading process is sped up. If the user doesn't have any interest in metadata at a certain time, they don't need to waste time waiting for metadata to be loaded into the program.

The second option – two stage sorting – is something slightly more sophisticated and tries to solve the problem of sorting large sets of images, which the client also mentioned. When two stage sorting is enabled, whenever the user chooses to sort more than 2000 images, they are sorted in two steps. First, they are matched to a saved similarity-trained map being a 50x50 grid of nodes. Because this map is trained earlier, the matching is done very quickly even with large sets of images. Images may overlap at this stage – each is matched to the best node. The user is then asked to select a subset of all these images to be sorted the usual way. Of course, if the user wishes to do so, they may also select all images. However, in most cases this two-stage approach will greatly speed up the sorting process, as the user will be able to choose a smaller subset that they are interested in several times, and get results very quickly. The idea of two stage sorting is illustrated below.

![Two stage sorting concept](image)

Figure 5.7.2-3 Two stage sorting concept

A picture showing how the images are matched to the beforehand trained map is visible below. Images may overlap each other – in that case only the topmost image is visible. Images are not so well sorted as if the map was trained individually for them, but it is still easy to select a region of interest and have it sorted the standard way.
Another thing that the client wanted was the possibility of cancelling time-consuming operation such as loading images and sorting large sets of images by similarity. This requirement was met by adding an 'Abort' button to the progress dialogs, as shown below.

Figure 5.7.2-4 Abort option in progress dialogs
When the abort button is pressed, the currently performed action is immediately terminated.

The last request that the client had was to enable showing hidden images quickly, as well as previous sorting states without the need to perform actual sorting once again. To achieve this, hidden images were decided not to be deleted from program memory, but just stopped from being displayed. Also, whenever the images are resorted, their previous coordinates on the display panel are saved, so that the previous sorting stage can be shown immediately when the user requests this. The button ‘Back to previous image display’ added to the context menu provides this functionality.

![Figure 5.7.2-5 Return to previous display option](image)

After all the improvements requested by the client were made, the application was once more tested to ensure that all options work correctly. This testing was separately performed by several different people. Once the results were satisfactory, the final product was presented to the client.
6. Evaluation

A good evaluation is essential to prove that the whole project was useful and not simply a waste of time. The evaluation should refer to both the project in general and the project’s outcome – the final product. The latter can be split into two parts done by different people. One type of product evaluation is done by the client, whereas a second, different type of product evaluation should be done by the developer themselves.

6.1. Critical Evaluation of the Final Product by the Client

All the client’s primary requirements were gathered when the initial agreement was prepared, and are mentioned in section 2.1.2. Every one of those requirements was fully met.

As to the additional requirements conveyed as feedback, nearly all were met completely, whereas some were met partially. The requirements that can be specified as only partially met are those concerning the loading and sorting speeds. Unfortunately, improving those speeds is at a certain point limited by hardware, and this barrier cannot be overcome.

Nonetheless, the client is fully satisfied with the final product and expresses his gratitude to the developer. A copy of the critical evaluation that the client provided after seeing the final product is attached as Appendix B-4. The general conclusions that can be made based on this document are as follows:

- All the initial requirements and in general also additional requirements were met
- The client is satisfied with the final product and expresses his gratitude to the developer
- There are only two issues which the client finds worth considering for improvement: image loading speed and application graphics

The last point mentioning areas of potential improvement can be used for any future development of the created application. This will be covered in more detail in the next chapter.

6.2. Comparison of the Final Product with Existing Software

Satisfying the client’s requirements is the most important objective of the developer. However, in order to properly evaluate a project, its outcome needs to be critically assessed in terms of usefulness and novelty that it introduces. In order to do this, the final product should be compared with other similar software.

As it was mentioned at the very beginning of this report, there are hundreds of programs capable of sorting images. However, they do not perform the same kind of sorting – they do not offer similarity sorting. Therefore, a comparison with those programs, or even with programs such as Picasa are difficult. Image Organizer clearly offers something new and different, and as such has to be recognized as an innovative and useful tool.

There is only one program, at least to the best knowledge of the developer, which is capable of performing similarity sorting the way the Image Organizer does it. This program is a prototype called Image Sorter v3 (University of Applied Sciences – Berlin 2008). The features of both applications have been compared in a table for better clarity. The testing was
performed on the hardware mentioned at the beginning of chapter 5 (Intel Core 2 Duo 2.4 GHz, 3 GB RAM)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Image Organizer</th>
<th>ImageSorter v3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional sorting</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Visual similarity sorting</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Editing metadata describing images</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Filtering images according to metadata</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Loading speed of 200 large images*</td>
<td>13.4 s</td>
<td>9.9 s</td>
</tr>
<tr>
<td>Loading speed of 10000 small images**</td>
<td>44.1 s</td>
<td>21.3 s</td>
</tr>
<tr>
<td>Sorting speed***</td>
<td>1.8 s</td>
<td>1.2 s</td>
</tr>
<tr>
<td>Large image set sorting speed**</td>
<td>2-stage sorting, makes sorting 10000 images easy and quick</td>
<td>traditional, around 92 seconds</td>
</tr>
<tr>
<td>Sorting quality* (subjective)</td>
<td>Very good</td>
<td>Very good</td>
</tr>
<tr>
<td>Operations on selected images</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Easy zooming</td>
<td>Yes, fully smooth thanks to separate thread</td>
<td>Yes, not fully smooth</td>
</tr>
<tr>
<td>Easy image selection and operations on selected images</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Potential program portability</td>
<td>Yes</td>
<td>No data</td>
</tr>
<tr>
<td>Graphics</td>
<td>Good</td>
<td>Very good</td>
</tr>
<tr>
<td>Adding new images to those already loaded into the program</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Adding single images without resorting everything</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Additional options</td>
<td>Several</td>
<td>Numerous</td>
</tr>
</tbody>
</table>

Table 6.2-1 Image Organizer and Image Sorter comparison

* A test set of 212 personal camera photos (4 Mega Pixels) was used
** A test set of 9908 images was used, attached as part of additional materials in folder TestSet_2 on the accompanying CD.
*** A test set of 1316 images was used, attached as part of additional materials in folder TestSet_1 on the accompanying CD.

Although Image Organizer and ImageSorter have a lot in common, each offers a lot of different options. Image Sorter actually achieves slightly better image loading and sorting speeds, and in general offers more functions. However, a number of new solutions implemented in Image Organizer allow it to outclass ImageSorter in several aspects, even though ImageSorter is piece of software that has been developed over several years. The most important of these aspects is sorting tens of thousands of images. Two-stage sorting makes sorting such large sets easy with Image Organizer. Another aspect which was not available in ImageSorter but is available and very handy in Image Organizer is metadata support. A further advantage offered by Image Organizer is the possibility of adding new single images without resorting all existing images.

As a conclusion, it can be said that Image Organizer is a unique piece of software which cannot be substituted by other programs. And most importantly, it explores a new, only emerging field of computer science – content-based image sorting.
6.3. Fulfilment of Project Objectives

Apart from creating a useful final product the project also had other objectives, listed in section 1.2. A brief overview of how these were met is presented below.

1. Requirement gathering

It is believed that all requirements were gathered well and made clear – both the initial requirements and the additional requirements that the client provided as feedback.

2. Similar product analysis

The performed analysis of similar software was extensive and provided some valuable insight into the currently available solutions. Many ideas used in existing software were used and therefore a lot of time has been saved.

3. Background research and justified technology choice

The choice of C++, the QT library and the Eclipse platform is believed to have been a very good choice, matching the technology to the type of faced problem very well.

4. Prototype design and implementation

It is believed that the design stage was treated with the necessary attention and concentration that it requires. This in turn allowed to greatly speed up the implementation process, which successfully yielded the first prototype of the product.

5. Feedback gathering and prototype improvement

Close cooperation with the client ensured that all requests and additional requirements that the client had were analysed and implemented as described in chapter 5. This way the final product could be adjusted to client’s exact needs.

6.4. Summary

The project can be assessed as successful because of the following:

- All the client’s primary requirements and a great majority of secondary requirements were fully met
- All project objectives were completed
- Coventry University requirements were met
- The developer gained new skills and valuable experience throughout the development process

As it eventually turned out, the topic of the project was suitably chosen, making its completion as part of a Bachelor’s Final Year Project feasible.

The fact that the client is content with the created final product and the novelty that the product introduces in the computer science field make the whole undertaken effort worthwhile and give credit to the developer.
7. Conclusions

7.1. Achievements

The achievements of this project include:

- Meeting the client’s and Coventry University objectives (as analysed in chapter 6)
- Creating this Final Report
- Creating a piece of software – Image Organizer

It is believed that the above is exactly what has been the purpose of the project from the very beginning. Therefore, the developer may be content with the final outcomes of the project.

7.2. Possible Future Work

As Image Organizer introduces a new concept of image sorting – according to visual similarity, a lot of research can be done in an attempt to find ways of doing this more efficiently. Testing different variations of methods implemented in Image Organizer can perhaps yield better final results. However, this process would certainly require a lot of time from a developer attempting it.

Apart from similarity-sorting efficiency issues, the following improvements could be implemented in Image Organizer:

- Creating a more ‘aggressive’ JPEG decoder, which would enable loading images in low quality more quickly than the default Qt decoder
- Creating a database that would store miniatures of images previously loaded into the program – this will make loading them again much faster
- Making the application fully portable by replacing windows-specific file functions with portable Qt functions (this shouldn’t require a lot of work so whenever the need arises it may be done)
- Improving graphics to make the application more pleasurable to the eye
References


available from <http://davis.wpi.edu/~matt/courses/soms/>
[15 September 2008]

Google (2009) *Picasa Basics: Picasa 3 feature overview* [online]
available from <http://picasa.google.co.uk/support/bin/answer.py?answer=93773>
[16 October 2008]

[27 October 2008]

Kohonen, T. (2005a) *SOM Toolbox: Intro to SOM by Tuevo Kohonen* [online]
available from
[27 October 2008]

available from
[27 October 2008]

Retrieval with MPEG-7 Content Descriptors, *IEEE Transactions on Neural
Networks*, Vol. 13, No. 4, July 2002, pp. 841-853

Lux, M.(2007) *SemanticMetadata.net >> Caliph and Emir* [online]
available from <http://www.semanticmetadata.net/features/>
[30 October 2008]

Maas, C. (2003) *Freeware Hex Editor XVI32* [online] available from
< http://www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm>
[19 January 2009]

Modi, V. (2009) *Color Descriptors from compressed images* [online]
available from
[10 January 2009]

MPEG (2002) *Multimedia content description interface: Part 8 – Extraction and Use of
MPEG-7 Descriptors*, ISO/IEC TR 15938-8:2002

Software*, ISO/IEC IS 15938-6:2003
also available online at
<http://tiger.ire.pw.edu.pl/asm/lab/lab5/MPEG7-XM.zip>
[15 September 2008]

MPEG (2008) *MPEG-7 Overview* [online]
[20 September 2009]


[10 October 2008]

[22 January 2009]

[22 January 2009]

[10 October 2008]

[20 September 2008]

[22 September 2008]
Bibliography

BioElectroMech (2005) JPEG Header Format [online]
available from <http://www.fastgraph.com/help/jpeg_header_format.html>
[12 October 2008]

Engelschall, Ralf S. (2005) All about GIF 89a [online]
available from
<http://netsw.org/graphic(bitmap/formats/gif/gifmerge/docs/gifabout.htm>
[14 October 2008]


Shoukry, L. H. (2009) Visualizing the MPEG-7 Color Layout Descriptor [online]
[25 January 2009]

[20 September 2008]


Wikipedia (2009a) HSL and HSV [online] available from
<http://en.wikipedia.org/wiki/HSL_color_space#Conversion_from_RGB_to_HSL_or_HSV> [03 January 2009]

Wikipedia (2009b) JPEG [online]

available from <http://www.qtforum.org/> [02 December 2008]
Appendix A – Project Brief
Appendix B – Cooperation with Client

B-1. Initial Agreement

Similarity-Based Image Sorting Application

Client’s Requirements – Based on Interview

The Interview has been performed on the 28.12.2008 with my client Mr Piotr Petraš. The general conclusions are presented below.

1. The needs that the product should satisfy.

The finished product should be a program designed to operate with large sets of images. The application should enable displaying images in an organised way. Apart from that, the final program should enable easy retrieval of an image when no query image is available. This can only be achieved when all displayed images are sorted by visual similarity – thus enabling the user to quickly find what they are looking for. The user should also be able to choose an image from within the program and perform some basic operations on that image, such as changing its location on the hard drive or editing the keywords describing it.

2. The input data for the application.

The type of images that will most often be used is the JPEG format. The application does not necessarily have to support other formats, but it would be an additional bonus if it did.

3. The options that the GUI of the application should offer.

The application should display all the chosen images in an easily accessible way. The user should be able to magnify any chosen region of all the images. It should also be possible to select a whole group of images and delete them, move them to another directory, edit keywords describing them or remove them from being displayed. It should furthermore be easy to access basic information about any image – such as name, date taken, resolution, location and size.

4. Additional requirements.

It would be appreciated if the speed and efficiency of image sorting were good. If it should however turn out impossible to achieve both, the speed should enable sorting a few thousand images (at once) in reasonable time.

Apart from visual similarity sorting, filtering images that match specified keywords is also required.

Date and client’s signature

PIOTR PETRAŚ CONSULTING
ul. Gwiaździsta 27/214
01-851 Warszawa
REGON: 140545984 NIP: 118-153-87-77
B-2. Feedback for Prototype 1

Similarity-Based Image Sorting Application

Client’s opinion after seeing prototype version 1

After developing the first prototype version of the product, it was shown to the client, Mr Pierr Ferras, and he was asked for feedback. His views are presented below.

1. Please state how you find the GUI of the application.
   - Excellent
   - Very good
   - Good, but with room for improvement
   - Satisfactory
   - Poor, but acceptable
   - Unsatisfactory

Comment.

The application makes a very good first impression. The tree-view of all the folders on the left makes it easy to navigate the data on the computer and the small miniature in the bottom left corner makes it easier to decide whether or not to add an image to the program. However, the tree-view does not display images in the best order – it should display folders first and single files later. Also case sensitivity is an issue – the sorting order should be case insensitive.

The display of all the images is also well thought out, as it occupies most space and makes it easy for the user to interact with the added images. However, a number of things could still be improved. Firstly, it should be possible to change the background colour behind the displayed images. Secondly, it would be better if the main window of the application started as maximized rather than at a fixed resolution – making things more visible. What is more, it would be useful if there was some place on one of the panels, where information about a selected image would be visible. This would make work a lot easier. If a miniature of any image that the mouse cursor hovers above appeared in the bottom left corner, it would be an additional bonus.

2. Please state how you find the functionality of the application.
   - Excellent
   - Very good
   - Good, but with room for improvement
   - Satisfactory
   - Poor, but acceptable
   - Unsatisfactory
Comment.

Adding images from any chosen folder as well as in a single fashion is a big merit, as it makes work a lot easier. Zooming with the mouse wheel is also impressive. So is the scrolling of all images after the sorting. However, a number of things should still be improved. Firstly, it should be possible to select any number of images, and not only one-by-one while double-clicking, but for instance by selecting all images in a drawn rectangle. It should also be possible to perform multiple operations on the selected images – apart from the options currently available also copying them and opening them in a graphics editing program, for example paint. Also hiding images, then resorting the remaining ones without those hidden would be appreciated. Another thing that could be added to the existing option of similarity sorting is sorting by name, size and date taken. These options might be required in future so their presence would be an advantage.

3. Please give your overall impression of the application.
   - Excellent
   - Very good
   - Good, but with room for improvement
   - Satisfactory
   - Poor, but acceptable
   - Unsatisfactory

Comment.

The application seems to be on the right path to becoming a very useful tool. What remains to be improved, apart from the above mentioned, are issues regarding usability. It would be good to have a progress dialog showing when time consuming operations such as loading images and sorting them, will finish. Also adding single images without the need to resort everything would be an additional advantage. And if possible, it would be greatly appreciated if the speed of loading images and sorting them could be improved.

Date and client’s signature: Niewai 18/02/2009

PIOTR PETRAS CONSULTING
ul. Gwiazdzista 27/214
01-651 Warszawa
REGON: 140645864 NIP: 118-153-87-77
B-3. Feedback for Prototype 2

Similarity-Based Image Sorting Application

Client’s opinion after seeing prototype version 2

After developing the second prototype version of the product, it was once again shown to the client, Mr Piotr Fietić, and he was asked for feedback. His views are presented below.

1. Please state how you find the GUI of the application.
   - Excellent
   - Very good
   - Good, but with room for improvement
   - Satisfactory
   - Poor, but acceptable
   - Unsatisfactory

Comment.

The GUI is very good. Information about loaded images is easily accessible, the bottom panel with extra information is a great idea and the top toolbar with available sorting options, as well as zooming and clearing images is good. Keyword filtering is also quite well thought out, but it could be made more visible if all the optional selection fields vanished when they aren’t needed – then they wouldn’t attract attention when it is unnecessary. Information about an image that appears when the mouse cursor hovers above it is very useful, but it should vanish as soon as the mouse stops hovering above the image, otherwise it is confusing.

2. Please state how you find the functionality of the application.
   - Excellent
   - Very good
   - Good, but with room for improvement
   - Satisfactory
   - Poor, but acceptable
   - Unsatisfactory

Comment.

Most importantly, the improved speed of loading images and sorting them is a big advantage. However, it would be an advantage if it were possible to specify whether or not to load images from subfolders.

The new options of sorting according to name, size and date work very quickly and make a good impression. Selecting images and performing different kinds of operations on them is very easy. Although when hiding images, they should not be deleted from memory as at the moment – reloading them takes too much time. It would be better if they were still kept
in RAM and simply stopped from being displayed. The progress dialogs are good, but should include the option of cancelling a process – if for example the user decides that too many images have been loaded and the sorting will take too long, it should be possible to stop it. Another thing that should be added is the possibility of returning to a previous state in the program. When the user sorts images by similarity, and then resorts a subset, it should be possible to quickly go back to the sorted display of all images quickly, without the need to sort them once again. The order should be somehow stored in memory.

3. Please give your overall impression of the application.
- Excellent
- Very good
- Good, but with room for improvement
- Satisfactory
- Poor, but acceptable
- Unsatisfactory

Comment.

The program make a very impression and has many useful features. Some of the available options are very interesting and will surely make organising images much easier for anyone willing to make use of it. The only thing that could be still be made better is the image loading speed, and the sorting speed in case of large image sets. However, even at present, these speeds are acceptable. The GUI is clear and easy to use. So recapitulating, I am looking forward to seeing the final version of the product with the minor improvements that it still requires.

Date and client’s signature: Piotr Pétraś Consulting

Piotr Pétraś Consulting
ul. Gwiazdowa 27/213
01-101 Warszawa
REGON: 140645864 NIP: 118-153-87-77
B-4. Evaluation of Final Product

Similarity-Based Image Sorting Application

Client’s evaluation of final product

The final version of the application has been developed based on the initial interview, and the feedback that the client, Mr Piotr Fetrai, kindly provided after seeing the first, and later the second prototype version. Like before, the client was asked to express his opinion after seeing the final version. His views are presented below.

1. Please state how you find the GUI of the application.
   - Excellent
   - Very good
   - Good, but with room for improvement
   - Satisfactory
   - Poor, but acceptable
   - Unsatisfactory

Comment.

The GUI is very good. Although it might lack some artistic design and sophisticated graphics pleasing for the eye, it is clear, easy to use and makes an overall very positive impression. All interactions with the application are performed in a smooth manner and so I am fully satisfied with this aspect.

2. Please state how you find the functionality of the application.
   - Excellent
   - Very good
   - Good, but with room for improvement
   - Satisfactory
   - Poor, but acceptable
   - Unsatisfactory

Comment.

The overall functionality of the program is impressive. All previously mentioned issues have been improved, and most importantly the sorting speed has been further increased. Approximated sorting of images and choosing subsets for a similarity sort in case of very large image sets are a very good idea, and completely solves the issue of sorting speed.
3. Please give your overall impression of the application.
   - Excellent
   - Very good
   - Good, but with room for improvement
   - Satisfactory
   - Poor, but acceptable
   - Unsatisfactory

Comment.

The only place where I can see room for improvement is image loading speed, although I understand that further image loading speed increase might not entirely lie within the developer's capabilities. Therefore, the option to choose whether or not to include keyword support and thus possibly increase image loading speed is a very good idea.

In general I am very happy with the final version of the application. I am sure that it will prove to be very useful and will make my work with digital images a lot easier.

Date and client's signature

Piotr Petras Consulting
ul. Gwiaździsta 27/214
01-851 Warszawa
REGON: 14064594 NIP: 118-193-87-77
Appendix C – Additional Materials on CD

...User manual including minimum hardware requirements.