CS3354 Software Engineering
Final Project Deliverable 2

ITSAMI
Image Tagging Software Applicable for Many Images

Group #: 10

Taufeeq Ali, Connor Ford, Fernando Hernandez, Bowen Liu, Tanner Raley, Asher Schubert, San Yun, Joel Zuniga

Delegation of Tasks

1st Deliverable:

- **Taufeeq Ali**: Class Diagram
- **Connor Ford**: Outlining Software Process Model, Functional Requirements, GitHub repository
- **Fernando Hernandez**: Use-Case Diagram
- **Bowen Liu**: Updated Version of Proposal
- **Tanner Raley**: Non-Functional Requirements, Architectural Design
- **Asher Schubert**: Activity Diagram
- **San Yun**: Class Diagram
- **Joel Zuniga**: Functional Requirements, Non-Functional Requirements

2nd Deliverable:

- **Taufeeq Ali**: Slides and Presentation
- **Connor Ford**: Project management, GitHub repository, Conclusion, References
- **Fernando Hernandez**: Software Testing
- **Bowen Liu**: Similar design comparisons
- **Tanner Raley**: Project Scheduling, Cost, Effort and Pricing Estimation, Project Duration and Staffing, Estimated Costs of Hardware, Software, and Personnel
- **Asher Schubert**: Refinement of Requirements, UI Mockup
- **San Yun**: Slides and Presentation
- **Joel Zuniga**: UI

Project **Deliverable 1 Content: (Note: with all corrections based on feedback)**

1. **Motivation**

   Digital collage art involves local collection of thousands of images with varying styles. Artists have a lot to choose from between blueprints, text documents, 3D renders, even corrupt program outputs. A system to automatically tag and search through local archives of images already exists in the form of a command-line project called rclip by Yurij Mikhalevich but is based on the aging OpenAI CLIP model. A new system could be developed that utilizes a GUI for viewing images based on search queries and runs on a newer pre-trained image tagging model.

2. **Goals:**

Our goal is to develop an intuitive and efficient image tagging software that leverages a modern pre-trained image tagging model, offering a graphical user interface (GUI) for seamless navigation and search through large local archives of diverse images. By improving upon existing solutions like rclip, we aim to provide artists and users with a more user-friendly and powerful tool to automatically tag, organize, and retrieve images based on descriptive queries.

3. **Delegated Tasks by Members**

a. Frontend (GUI)
-Asher Schubert
-San Yun
-Joel Zuniga
b. Backend (Model Interface)
-Taufeeq Ali
-Connor Ford
c. Backend (GUI Interface)
-Fernando Hernandez
-Bowen Liu
-Tanner Raley

2) Repository Link

https://github.com/connor-ford/itsami

3) Delegation of Tasks

- **Taufeeq Ali**: Class Diagram
- **Connor Ford**: Outlining Software Process Model, Functional Requirements, GitHub repository
- **Fernando Hernandez**: Use-Case Diagram
- **Bowen Liu**: Updated Version of Proposal
- **Tanner Raley**: Non-Functional Requirements, Architectural Design
- **Asher Schubert**: Activity Diagram
- **San Yun**: Class Diagram
- **Joel Zuniga**: Functional Requirements, Non-Functional Requirements

4) Software Process Model

We will be using the Waterfall process model, as a linear, sequential approach with distinct, independent phases would suit the static nature of our project. By separating requirements, design, implementation, and testing into distinct phases of the timeline, we can ensure a better final product than we would by using an incremental or iterative process model.

5) Software Requirements

5.a) Functional Requirements

- The software shall allow users to load images and their tags from directories into the application.
- The software shall be able to display images and their tags.
- The software shall be able to process images and create tags with the AI model.
- The software shall be able to save an image's tags as metadata for later use.
- The software shall have a search function to find images by their tags.
- The model should be able to be used through a Command Line Interface (CLI) in addition to within the application.

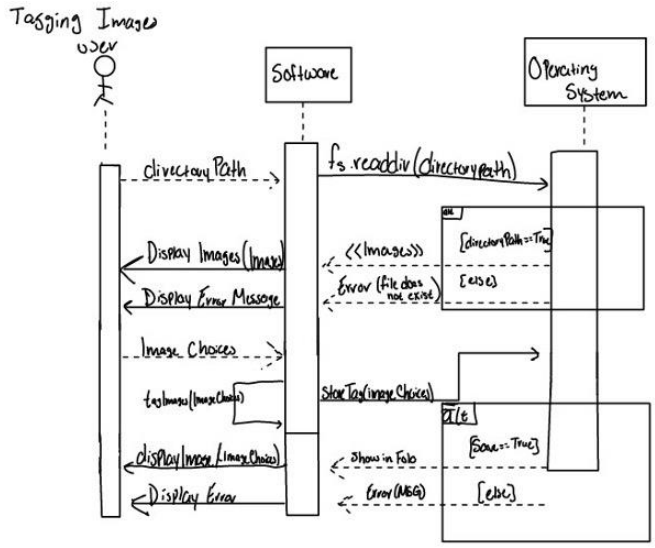
5.b) Non-Functional Requirements

- The application shall be able to access and save multiple images contained in directories to the user's system along with their associated tags.
- The application shall return an image tagging request within several seconds to some minutes at most depending on image complexity and size
- The application shall be able to support multiple tags per image.
- The application should be portable.

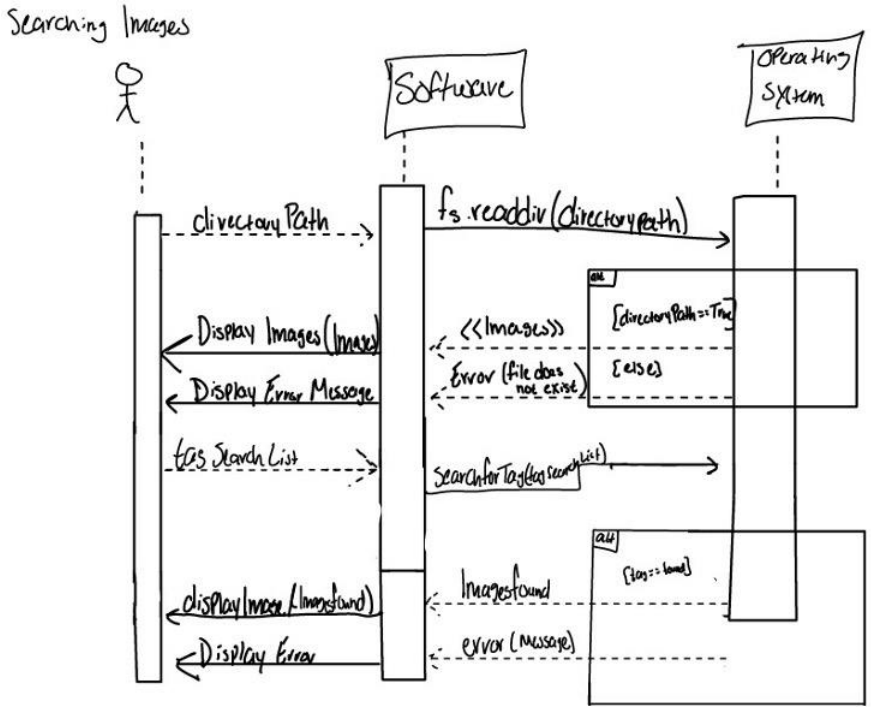
6) Diagrams

Sequence Diagrams

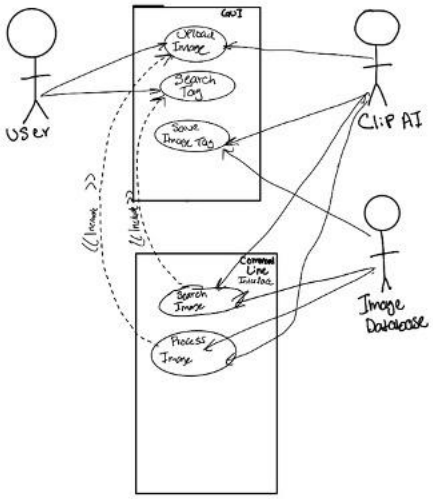
1) Tagging Images



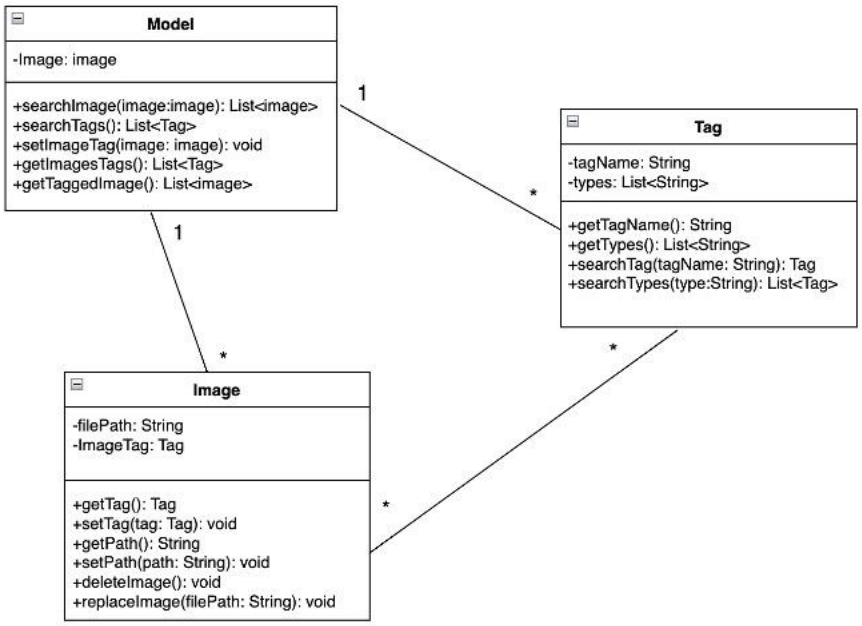
2) Searching Images



Use Case Diagram



Class Diagram



Class	Data	Method
Image	String filePath Tag imageTag	getTag() setTag() getPath() setPath() deleteImage() replaceImage(String filePath)
Tag	String tagName List<String> types	getTagName() getTypes() searchTag() searchTypes()
Model	Image Image	searchImage(Image) searchTags() setImageTag(Image) getImageTags() getTaggedImage()

7) Architectural Design

The architectural design pattern that our group has chosen to utilize is the Repository pattern. This pattern was selected because the system requires the application to generate large amounts of data (specifically, image tags) that must be stored for extended periods. In addition, the chosen pattern implements a structured method to handle data access, allowing for ease of adding, modifying, or deleting tags that have been associated with images.

Project Scheduling, Cost, Effort and Pricing Estimation, Project Duration and Staffing

1. Project Scheduling

Task	Effort (person-days)	Duration (days)	Dependencies	Activity
T1	10	4	N/A	planning code structure
T2	20	10	N/A	Culminate test image database

T3	45	15	T1	develop tagging software
T4	15	7	T2	Create possible image tag database
T5	15	5	T3	test individual tag creation
T6	5	3	T3	test new tag creation
T7	20	5	T3	test new image uploading and tagging

Weeks:	1-2	3-4	5-6	7-8	9-10	11-12	13-14
Taufeeq							
Connor							
Fernando							
Bowen							
Tanner							
Asher							
San							
Joel							
T1	T2	T3	T4	T5	T6	T7	

2. **Cost, Effort and Pricing Estimation (**Functional Point)

Function Category	Count	Complexity Type	Total Complexity
Number of User Input	50	Simple	150
Number of User Output	10	Average	50
Number of User Queries	50	Average	200
Number of Data Files	1	Complex	15
Number of External Interfaces	5	Average	35

Note: These values are estimated based on expected user interaction

Therefore, using the above table our Gross Function Point equals 450. Then using the Processing Complexity Adjustment algorithm, the PCA for our project is 1.18. Then by inputting this information into the Functional Point algorithm, we can calculate that to be 531.

Effort for this project can be calculated at a 107 person-week given the team productivity of 5 FP / person-week. This means the project's duration is 14 weeks given an 8-person team.

3. **Estimated cost of hardware products**

The hardware required for this project includes work devices for the team, likely individual laptops for each project member. An important part of testing is ensuring cross-OS compatibility, so having access to different systems either through physical hardware or through virtual machines is vital.

4. **Estimated cost of software products**

The main software utilized will be IDE's for building the project and GitHub for source control. Using any IDE, for instance Eclipse or Visual Studio, is sufficient for this project, and is available for free to use. For source control, we will use GitHub Teams plan, available for $4/user/month, making the total software cost $32 a month.

5. **Estimated cost of personnel**

Personnel required is the 8-member team for the three-week project timeline, and an estimated 1–2-person team for maintenance and supervision upon release.

Software Testing

Black Box Testing

#	Description	Input	Expected Output
1	Test for tagging dogs	Image of a dog	Tagged image as Dog
2	Test for tagging cats	Image of a cat	Tagged image as Cat
3	Test for tagging skyscraper	Image of a skyscraper	Tagged image as Skyscraper
4	Test for tagging house	Image of a house	Tagged image as House
5	Test for image	Random Image (jpeg, png, gif, etc.)	Input is an image
6	Test for non-image	Website, Video, etc.	Error Message

7	**Test for tagging person**	**Image of a person**	**Tagged image as person**
8	**Test for duplicate image**	**Image that's already been tagged**	**Not tagged, displays "image already tagged"**
9	**Test for multiple images**	**Set of random images**	**Each image tagged**
10	**Test for tagging food**	**Image of food**	**Tagged image as food**

White box testing

#	Description	Input	Expected Output (EO)	Actual Output (AO)	Passing Criteria	Test Result
1	**Test for tagging dogs**	**dog.png**	**<Dog>**	**TBD**	**EO=AO**	**TBD**
2	**Test for tagging cats**	**cat.png**	**<Cat>**	**TBD**	**EO=AO**	**TBD**
3	**Test for tagging skyscraper**	**empirestate.png**	**<Skyscraper>**	**TBD**	**EO=AO**	**TBD**
4	**Test for tagging house**	**home.png**	**<House>**	**TBD**	**EO=AO**	**TBD**
5	**Test for image**	**x.png**	**True**	**TBD**	**EO=AO**	**TBD**
6	**Test for non-image**	**road.mov**	**"Not an image"**	**TBD**	**EO=AO**	**TBD**
7	**Test for tagging person**	**me.png**	**<Person>**	**TBD**	**EO=AO**	**TBD**

8	**Test for duplicate image**	**dog.png**	**“Image already tagged”**	**TBD**	**EO=AO**	**TBD**
9	**Test for multiple images**	**Cat.png Dog.png**	**<Cat> <Dog>**	**TBD**	**EO=AO**	**TBD**
10	**Test for tagging food**	**pasta.png**	**<Food>**	**TBD**	**EO=AO**	**TBD**

TBD: To be determined

Similar Design Comparisons

1. **rclip**(Existing CLI Tool)

   Purpose: Searches local images using OpenAI CLIP embeddings via text queries.
   Strengths:

   - Fast, scriptable, and lightweight
   - Works offline after setup

Weaknesses:

- No GUI: Requires command-line knowledge.
- Outdated Model: Uses older CLIP(ViT-B/32), missing newer improvements.
- No Auto-Tagging: Only search, no persistent tagging system.

2. Eagle App

Purpose: Asset manager for designers
Strengths:

- Visual Browsing: Grid/column views, quick previews.
- Manual Tagging: Support folders, labels, and annotations.
- Extensions: Some AI plugins(but not deeply integrated)

Weaknesses:

- No Built-in AI Tagging: Relies on manual input.
- Proprietary: Not open-source software

3. Adobe Bridge

Purpose: File browser with metadata management for creatives.
Strengths:

- Metadata Filtering: EXIF, IPTC, custom keywords.
- Integration: Works with Photoshop/Lightroom.

Weaknesses:

- No AI: Tagging is manual.
- Bloated: Heavy resource usage.

4. Hydrus Network

Purpose: Advanced local media tagging.
Strengths:

- Powerful Tagging: Boolean search, wildcards, namespaces.
- Customization: Plugins for external AI.

Weaknesses:

- Steep Learning Curve: Overwhelming for casual users.
- No Native AI: Requires manual setup.

Feature Comparison Table:

Feature	rclip	Eagle	Adobe Bridge	Hydrus	Our System Goal
GUI	❌	☑	☑	☑	☑(Modern)
Auto-Tagging	❌	❌	❌	❌	☑(AI-Powered)
Manual Tagging	❌	☑	☑	☑	☑
Offline Support	☑	☑	☑	☑	☑
Advanced Search	❌	☑(Basic)	☑(Metadata)	☑	☑(AI + Filters)
Performance	☑	☑(Medium)	❌(Heavy)	❌(Slow)	☑
Cross-Platform	☑	☑	❌(Win/Mac)	☑	☑
Metadata Support	❌	☑	☑	☑	☑
Batch Processing	❌	☑	☑	☑	☑

Extensibility	✗	☑ (Plugins)	☑ (Scripts)	☑ (Plugins)	☑ (Plugins)

Conclusion

For Deliverable 2, our team concentrated on applying core software engineering principles to carefully plan out the ITSAMI project. We developed and refined key project artifacts, including requirements specifications, process models, architectural designs, scheduling plans, cost estimates, and detailed testing strategies. This exercise gave us the opportunity to systematically walk through every major phase of the software development lifecycle, following a structured and disciplined approach.

Throughout the process, we ran into several challenges that are typical when tackling early-stage project planning. Crafting clear, realistic functional and non-functional requirements took multiple revisions to ensure everything aligned with our bigger project goals. Estimating cost and effort without a working implementation meant we had to make careful assumptions and extrapolations. Maintaining consistency between design artifacts, such as diagrams and architectural models, demanded ongoing communication and frequent changes. Lastly, creating a prototype of what the ITSAMI project would look like presented conflicting information with previous design artifacts that had to be remedied.

One of the biggest takeaways from this deliverable was how critical early, thorough planning is for reducing risks and setbacks down the road. Working within a team environment also reinforced the need for clear role delegation and strong documentation habits. That said, coordinating across such a large group wasn't easy; scheduling meetings was a challenge and forced us to figure out more flexible forms of communication to stay in sync.

In summary, Deliverable 2 demonstrates a solid grasp of the software engineering process and sets a strong foundation for the potential future development of the ITSAMI application.

References

Mikhalevich, Yurij. "Using Text Queries to Look up Unlabeled Images: A Command-Line Search Tool Based on Clip." *ThinkMind(TM) Digital Library*, 26 June 2023, www.thinkmind.org/index.php?view=article&articleid=content_2023_1_20_60011.