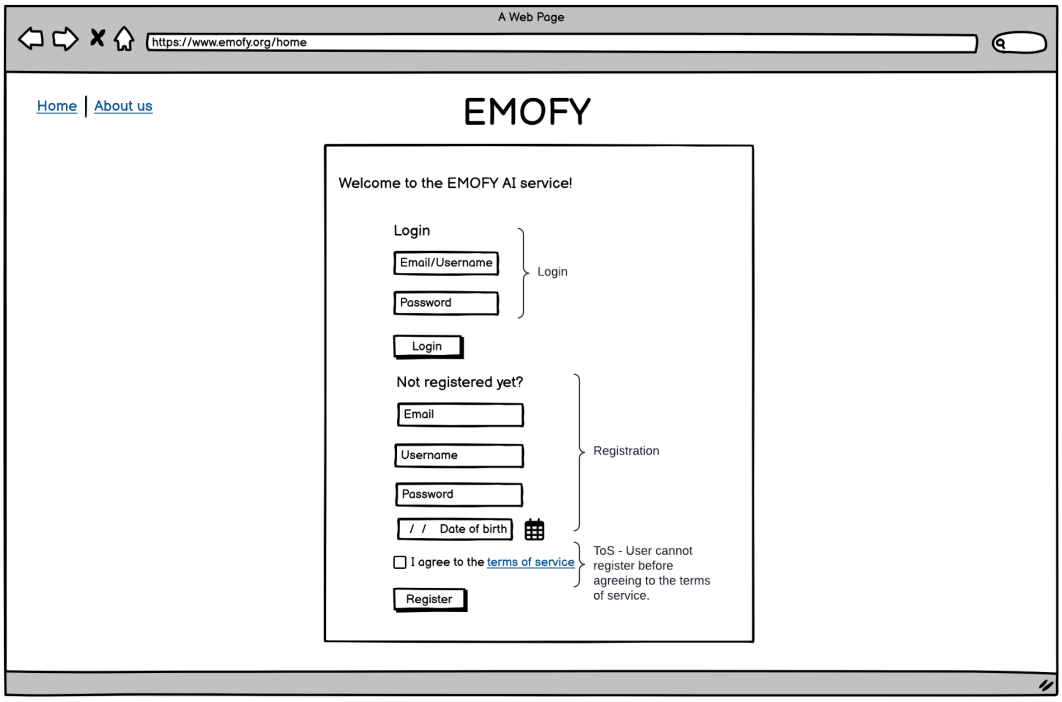
EMOFY Booklet

Alessio Maiola, Gabriele Matini, Placido Pellegriti

1 User Stories for EMOFY

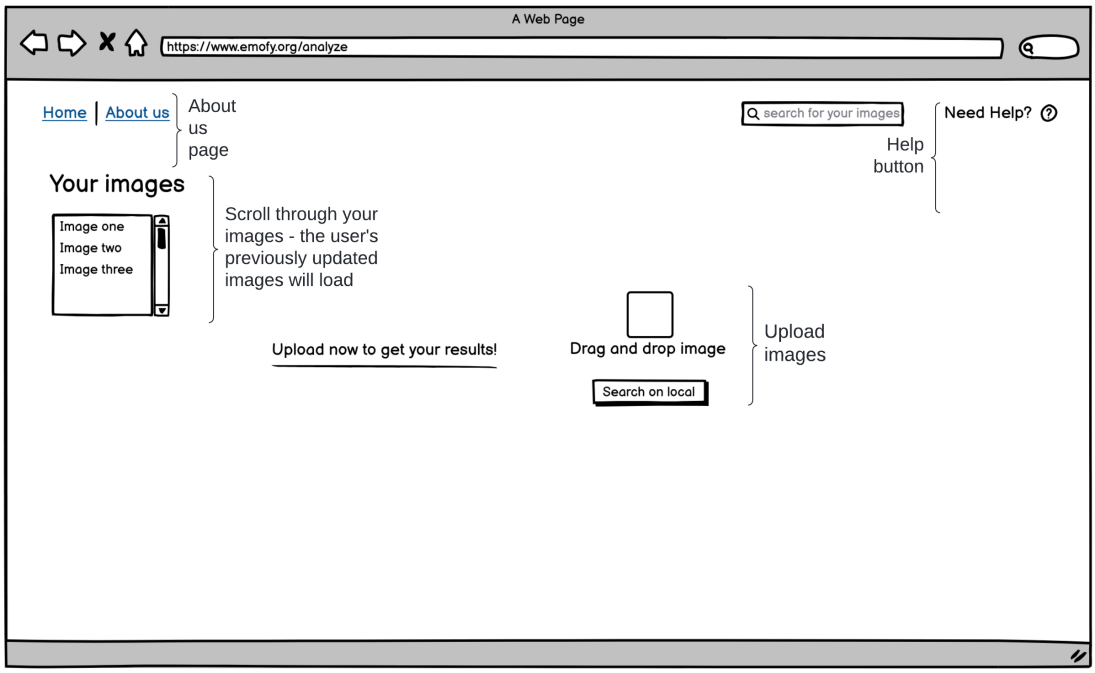
We'll see now all user stories of our microservices related to the mock-ups.

- 1. As a user, I want to be able to register, so that I have an account to use the services.
- 2. As a user, I want to be able to login, so that I can access the services.
- 3. As a user, I want to be able to read and accept the terms of service, so that I can be informed on how the data inserted by me will be treated.

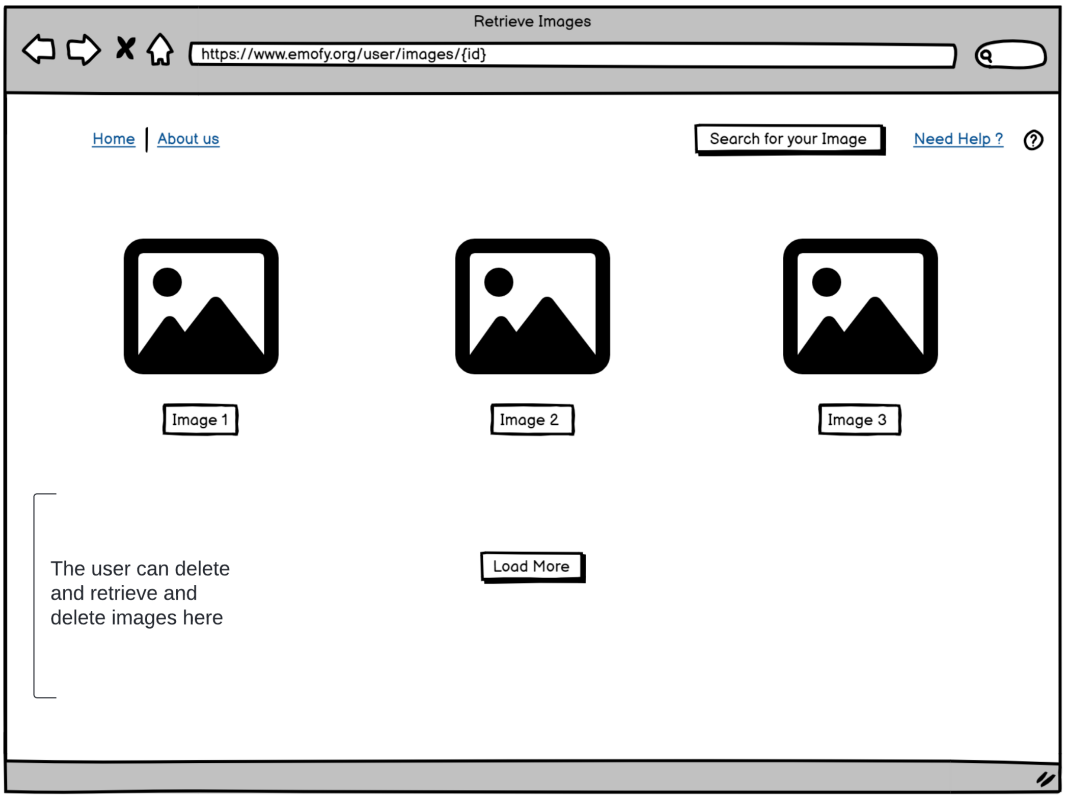


- 4. As a user I want upload and store images efficiently.

- 5. As a user, I want a UI that allows to efficiently use the transformations and the model, so I can use the offered services.
- 6. As a user, I want to click on the help button, so that I can be given information on how the site works.
- 7. As a user, I want to click on abous us button, so that I can be given information on the creators.
- 8. As a user, I want to use the ML algorithm on filtered pictures, to test the ML algorithm on different versions of the same picture

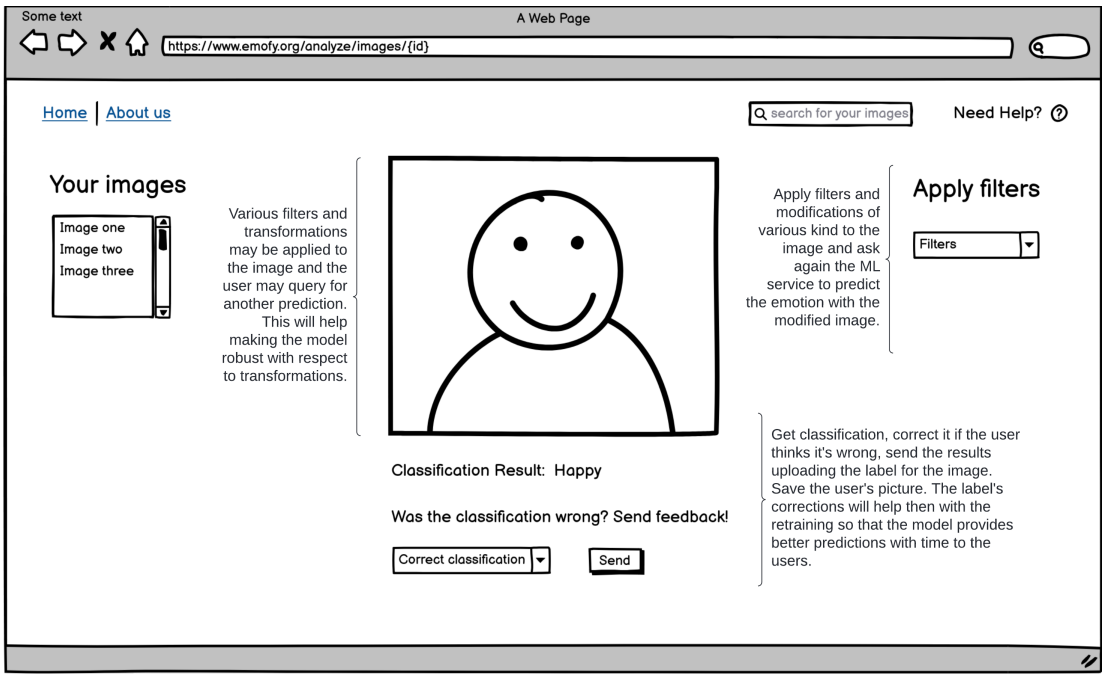


- 9. As a user I want the system to tell me if it can recognize a face in the picture I downloaded or not.
- 10. As a user, I want to be able to search for the images I uploaded in the past on the site, so that I can retrieve them and use them again.

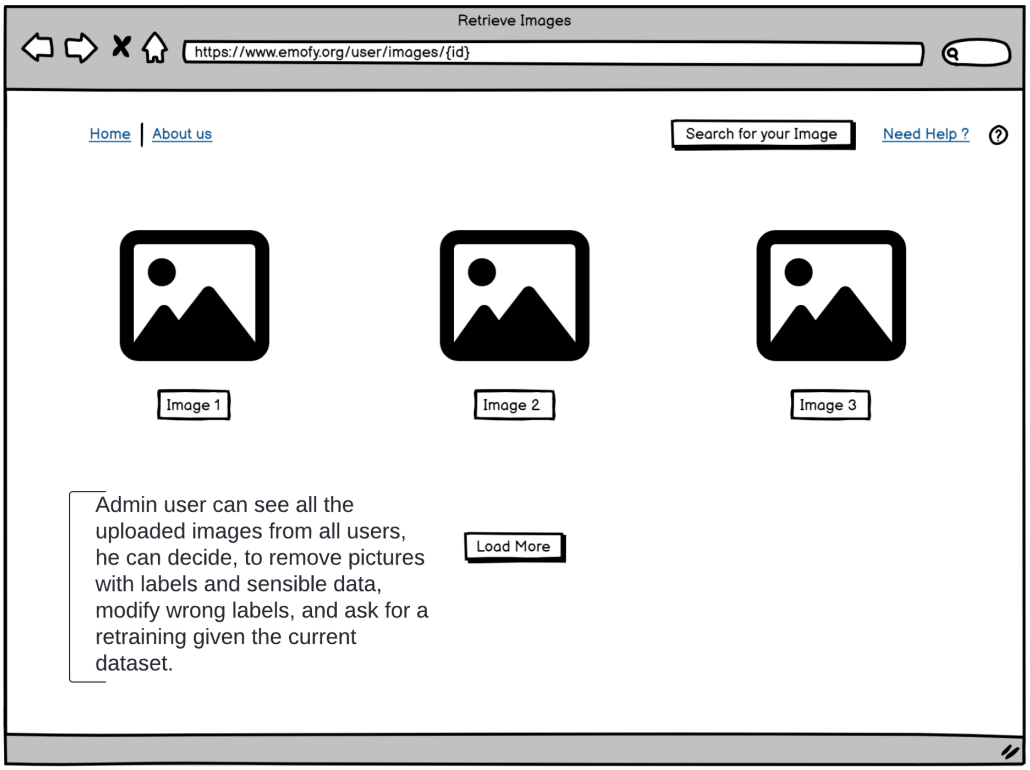


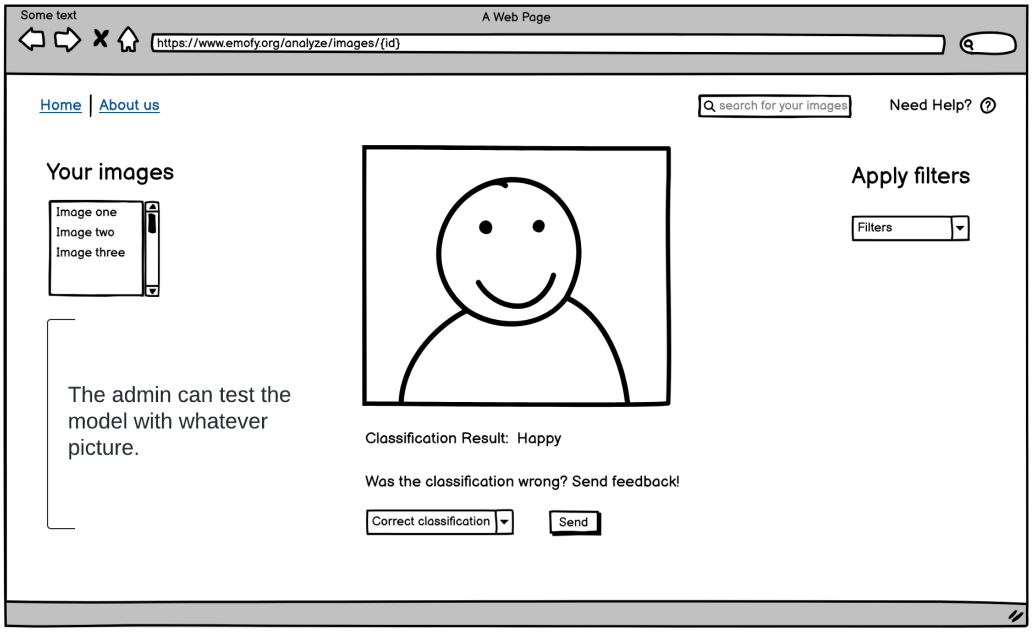
- 11. As a user I want upload images' labels, so they can be used for the AI model's retraining
- 12. As a user, I want to apply filters on the images I uploaded so I can see the result.
- 13. As a user, I want to store the result of Image processing, so I can download it later or test the model with it.
- 14. As a user, I want to apply multiple transformations on the images, so I can avoid to continuously store the result and upload the modified image.
- 15. As a user, I expect the predictions of the model to get better over time, as the quantity of collected images increases.
- 16. As a user, I want to query the ML model on the picture I uploaded, so that I can get the emotion recognition results.
- 17. As a user, I want the model to automatically find faces in the images I upload, so I can avoid resizing them.

- 18. As a user, I want to indicate the right class, if the classified one is not correct, so the model can improve its performance.



- 19. As a system administrator, I want to delete some users' insertions if they are wrong or they have sensible data and modify users' insertions, to take action with regard to users' misuse of the site
- 20. As a system administrator, I want to test the model on the present dataset to assess its effectiveness.
- 21. As a system administrator, I want to be able to see the model's performances when the model is re-training.
- 22. As a system administrator, I want to launch an automated retraining routine of the model.
- 23. As a system administrator, I want to retrieve stored images to train again the model. Non functional requirement: uploaded images will be used to train the model only if the user provided a feedback about the prediction, so noisy samples can be avoided.


Retrieve Images
https://www.emofy.org/user/images/{id}
Home | About us
Search for your Image
Need Help ?
Image 1
Image 2
Image 3
Admin user can see all the uploaded images from all users, he can decide, to remove pictures with labels and sensible data, modify wrong labels, and ask for a retraining given the current dataset.
Load More


Some text
A Web Page
https://www.emofy.org/analyze/images/{id}
Home | About us
search for your images
Need Help?
Your images
Image one
Image two
Image three
Apply filters
Filters
The admin can test the model with whatever picture.
Classification Result: Happy
Was the classification wrong? Send feedback!
Correct classification
Send

- 24. As a system administrator, I want to monitor what the users' behavior through logging, so that I can analyze what happens in my system and how it's being used (uses Kibana's interface).
- 25. As a system administrator, I want to to collect data from the other microservices and represent it in different ways (uses Kibana's interface).

2 COCOMO II estimation

We estimate now the effort expressed in man-months to deliver our application, as well as the total estimated time and the lines of code to develop it. We follow the COCOMO II model definition manual available for free on the internet.

2.1 Function Point estimation

We are using as reference for the estimates our function point estimation in UFP: the total is 61 UFP.

No.	Module	Function Name	Type	DET	RET / FTR	Complexity	FP
1	Users	Users	ILF	4	1	Low	7
2	Images	Images	ILF	4	1	Low	7
3	Users	Add User	EI	4	1	Low	3
4	Image Storage	Add Image	EI	5	2	Average	4
5	Users	Login	EQ	3	1	Low	3
6	Users	Update User Information	EI	3	1	Low	3
7	Users	Delete User Information	EI	1	1	Low	3
8	Image Storage	Retrieve All Images IDs	EO	5	2	Low	4
9	Image Storage	Delete Image	EI	1	1	Low	3
10	Image Storage	JWT-Token Service	EQ	1	1	Low	3
11	ML Model	Predict Image Emotion	EO	4	1	Low	4
12	Filters	Apply Filter	EO	4	1	Low	4
13	Image Storage	Retrieve Image	EQ	4	1	Low	3
14	Users	Login User	EI	3	1	Low	3
16	Image Storage	Update Label	EI	3	2	Low	3
17	ML Model	Detect Face	EO	3	1	Low	4

Table 1: Function Point Analysis Table

2.2 Lines of Code

We estimate the lines of code (in KLOC) using the following formula:

$$\text{KLOC} = \frac{\text{Function Points} \times \text{Language Factor}}{1000}$$

Where the language factor accounts for the language verbosity. Now, according to the language factor table, the average languages factors are:

Java 53
JavaScript 47
HTML 36

Table 2: Lines of Code per Function Point for the relevant languages KLOC/UFP

For Python no official estimate is present on the COCOMO II manual, although the consensus seems to be a factor of 10 or 20, so let us pick 15. Our Project is consistently made of Javascript, HTML, Python and Java, in percentages tracked by github, and these percentages are going to be used to weight the estimate too. According to our pre-project estimate, we thought that Python would take 30% of the project, since it's used for the AI, Java 40%, JavaScript 20% and HTML 10%. According to these estimates the KLOC cost was:

$$\text{Cost in KLOC} = \frac{61 \times 53 \times 0.4 + 61 \times 47 \times 0.2 + 61 \times 36 \times 0.1 + 61 \times 15 \times 0.3}{1000} = 2,36 KLOC$$

In reality our post-project data extracted from github percentages gives us a 37.1% for Java, 21.8% for Python, 15.1% for HTML, 14.8% for JavaScript and 11.2% for other languages, which makes it harder to compute, like CSS and Batchfiles. In practice, we will be using the exact amount of LOCs for these other files, since no empirical estimate is present. Below we have calculated with a tool the effective amount of Lines of Code, broken down by language: This

Language	**Files**	**Code**
Java	30	1187
Python	16	698
YAML	28	672
JavaScript	2	580
HTML	11	477
Bourne Shell	15	290
Maven	2	280
DOS Batch	15	253
CSS	2	223
XML	4	131
Dockerfile	5	51
SQL	1	41
JSON	2	8
Markdown	1	1
SUM:	**134**	**4892**

Table 3: Code Summary by Language

counting does not factor into the sum the blank lines or the comment lines. So let us consider the real percentages and compare the results with the actual 4k lines of code (not counting YAML lines and Maven lines).

$$\text{Cost in KLOC} = \frac{61 \times (53 \times 0.317 + 47 \times 0.148 + 36 \times 0.151 + 15 \times 0.218) + 672 + ...}{1000} = 3,930 KLOC$$

Circa 3 KLOC without Maven and Yaml.

The COCOMO estimate, even when helped by real GitHub percentages, would still be underestimating the cost by a margin of 1 KLOC. Clearly, the real cost in KLOC is expressed by the SUM field in the table cited above. In

conclusion the size estimate of our project was 2.36KLOC, but in reality the cost amounted to more or less 4KLOC (not counting some configuration YAML and Maven files).

2.3 Effort Estimation

The effort is expressed in person-months (PM) and is calculated as:

$$PM = A \times Size^E \times \prod_{i=1}^{n} EM_i$$

Where A is a constant (2.94), Size is the KLOC size we estimated and every Mi is an effort multiplier. E is an aggregation of five scale factors: PREC (precedentness), FLEX (flexibility), RESL (risk/architecture resolution), TEAM (team cohesion) and PMAT (process maturity). E is calculated as:

$$E = B + 0.01 \sum_{j=1}^{5} SF_j$$

where B is a constant 0.91 and SF_j is a scale factor. Let us now argue the scale factors: PREC is 6.20 (very low, the least experienced value), since the team was completely inexperienced to the technologies we had to use, and in fact, learning to use them was what took the most time and effort. FLEX is at 2.03 (high), since both in the requirements and in the technologies the team had choice, although some fixed points were kept unchanged throughout the project (AI usage, need for a login service and so on...). RESL is 4.24 (nominal), since there never were many risks involved in the project, but at the same time parts of the software architecture remained uncertain for a long time. TEAM is 1.10 (very high), since the TEAM never had major problems with communication and work overall. PMAT of course depends from the CMMI levels as well as key process areas, so our team could never have an incredibly high amount of process maturity. However, consistent usage of DevOps tools like Jenkins, GitHub and Docker, guaranteed at least some level of quality and readiness of the product. Our final PMAT is 6.24 (low): many key process areas, which make the PMAT level, are on the "rarely" scale or on the "doesn't apply" and "don't know" scale. They were not reported for the sake of brevity. Our final E value thus is:

$$E = B + 0.01 \times (6.2 + 2.03 + 4.24 + 1.1 + 6.24) = 1.1081$$

Now we have to calculate the effort multipliers. We will use the early design model to calculate first development cost. To this purpose the table on the next page has been provided.

Multiplier	Description	Value	Argument
PERS	Personnel capabilities	Low (1.26)	the team is inexperienced with many of the technologies but is very good at solving problems and fixing bugs and issues.
RCPX	Product reliability and complexity	Nominal (1)	the product is somewhat complex, but a lower need for reliability drives down the multiplier, since a bug is probably causing a minor inconvenience for a user and not a life threatening risk.
PDIF	Platform difficulty	low (0.87)	the software runs on the JRE, on the OS on Python and generally speaking on platforms that don't need many updates (low volatility), at the same storage and cpu constraints grow with user demand, which is not expected to be incredibly high.
PREX	Personnel Experience	Nominal (1)	the team is inexperienced with all the DevOps tools and frameworks at the start of the project, although languages like Python, Java and JavaScript are well known to the team.
FCIL	Facilities, support tools and multisite working	Very High (0.73)	the team is not in almost any need of multisite support since the members are all located in the same city and area, at the same time the usage of DevOps tools guarantee a robust support tool chain that enables an integrated lifecycle support.
RUSE	Reusable components	Very Low (0.7)	No components are intended for reuse.
SCED	Required development schedule and its strictness	high (1, with 130% of the optimal schedule being followed)	the team had a lot of time at its disposal.

Table 4: Code Summary by Language

Now that all cost factors are available, we can finally calculate the effort with CoCoMo II:

$$PM = A \times Size^E \times \prod_{i=1}^{n} EM_i = \text{ 3.5 man-months}$$

where size is 2.36KLOC, E is 1,081 and A is 2.5 for early design estimation.

Time to Develop (TDEV)

We can finally calculate the TDEV, with C = 3.67, B is 0.91, D is 0.28. These constant values are, as always reported on the COCOMO model definition manual.

$$\text{TDEV } (\textit{Time to Develop in months}) = C \times (\textit{Effort})^F \times \frac{SCED\%}{100}$$

With

$$F = D + 0.2(E - B)$$

$$F = 0.28 + 0.2(1.081 - 0.91) = 0.3142$$

$$TDEV = 3.67(3.5)^{0.3142}\frac{130}{100} = 7 months$$

3 Application of SCRUM in EMOFY Project

The EMOFY project adopted the SCRUM methodology to manage the development process, allowing for an iterative and adaptive approach. The focus was on building a microservices-based system that could handle various tasks such as user management, image processing, and machine learning model integration. By organizing the work into sprints, we were able to tackle complex requirements in a structured way, ensuring continuous progress and flexibility.

The project was divided into two sprints. Sprint 1 focused on establishing the core functionalities of the system, including user registration, image storage, and initial image processing capabilities. Sprint 2 aimed to enhance the system with advanced features like ELK stack integration, a user interface, and machine learning capabilities.

Given the diverse schedules among the team members, we needed to manage our time effectively. The team met regularly on weekends, typically on Sundays, to synchronize our efforts and discuss progress. This approach allowed us to maintain cohesion within the team and ensure that we met our sprint goals, despite the varying timelines.

3.1 Sprint 1: Establishing the Core

In the first sprint, our primary goal was to lay the foundation for the system. This included setting up essential services like user registration and login, image storage, and initial image processing capabilities. The team worked on implementing the login and registration service, integrating it with a user database and Google's API for authentication. Simultaneously, we developed a robust image storage service using MinIO, which included creating a CRUD API for managing image data and establishing communication between services.

This sprint also saw the introduction of basic image processing features, such as applying filters and transformations. By the end of Sprint 1, the core functionalities were in place, ready to support the more advanced features planned for the subsequent sprint.

The Burndown Chart for Sprint 1, shown in Figure 1, illustrates the progress over time. The chart indicates a steady decrease in the remaining effort, with most of the work completed towards the end of the sprint. This steady decline reflects consistent work and timely completion of tasks.

Table 5: Sprint Hours Allocation

PB Name	Task Name	Check Out	Remaining Effort [hr]
Sprint 1			
Login and Registration	Implement login and registration service	Gabriele	10
	Implement Users DB	Gabriele	6
	Implement Google API access	Gabriele	5
	Implement terms of service acceptance	Gabriele	1
Image Storage	Implementation of MinIO endpoint and Communication Setup	Alessio	11
	Implement Image CRUD API	Alessio	13
	JWT-based communication with login service and file preprocessing	Alessio	8
	Testing	Alessio	6
Image Processing	Implement filter service	Placido	5
	Implement help functionality	Placido	5
Sprint 2			
ELK Stack	Deployment and Internal Orchestration of ELK Stack service	Gabriele	18
	Connect ELK Stack service to all other services	Gabriele	4
	Add logging endpoints in all the other services	Gabriele	3
User Interface	Implement main page with orchestration of the microservices	Alessio	7
	Implement images page with preview of all the images	Alessio	2
	Implement about us and help pages	Alessio	2
	Integrate all the API calls	Alessio	3
Use ML Model	Training initial model	Placido	3
	Implement AI model serving	Placido	8
	Implement loading of latest available model	Placido	2
Retrain ML Model	Retrieval of new images from Image Storage	Placido	2
	Implement retraining logic	Placido	4
	Upload model to storage	Placido	1
	Creation of Docker Image for Model Serving	Placido	3
	Creation of retraining (cronjob) image	Placido	2

3.2 Sprint 2: Enhancing and Integrating

Building on the foundation laid in Sprint 1, the second sprint focused on integrating advanced features and refining the system. One of the key achievements was the deployment and orchestration of the ELK stack, which provided a tool for monitoring and logging. This allowed us to connect the various microservices and ensure that all interactions were properly tracked and analyzed.

The user interface (UI) was another major focus during this sprint. We designed and implemented a UI that integrates with the microservices, providing users with functionality such as image previews and support pages.

Moreover, this sprint marked the integration of machine learning capabilities into the system. We trained the initial model, implemented AI model serving, and set up a retraining mechanism that automatically updates the model based on new data. These advancements brought EMOFY closer to its goal of providing a responsive and intelligent system for emotion recognition in images.

The Burndown Chart for Sprint 2, shown in Figure 2, demonstrates effective progress management. The chart reveals again consistent reduction in remaining effort, reaching zero by the end of the sprint.

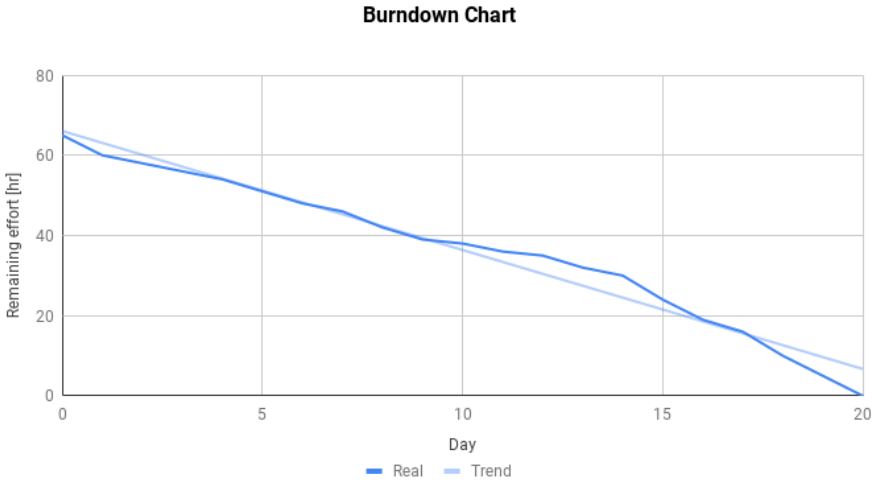


Figure 1: Burndown Chart for Sprint 1

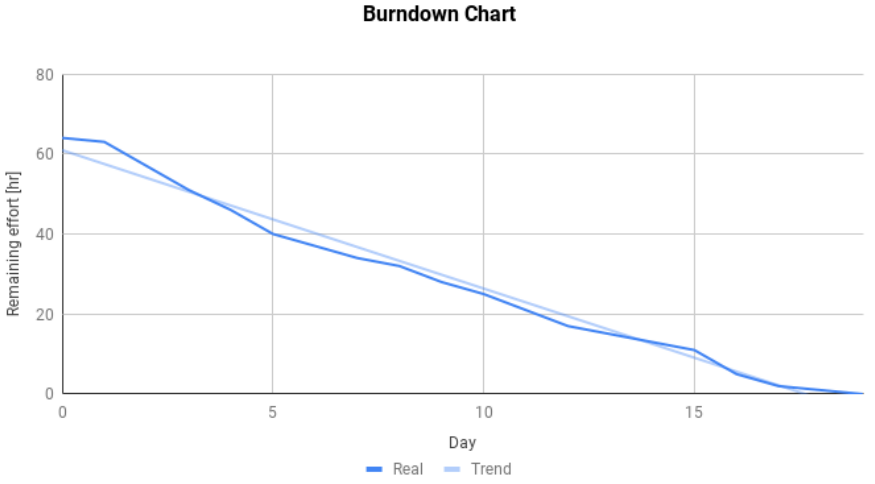


Figure 2: Burndown Chart for Sprint 2