Development Process

Mechatronics & Software

Team 3 - EasyRead: Instant Translation with Smart Wearable Device

Ding (Chris) Hao
Yimin (Jane) Pang
Yucheng Yao
Wenyu (Winnie) Yin
Taoming Yu
Xiang (Shawn) Zhang

Date	Developers	Change
Oct 02, 2024	Group 3	First version of the document

Table 1: Revision History

1 Overall Process Workflow[1]

Step	Agenda	Input	Output / Goal
1	Conduct hazard analysis, safety regulations, privacy rights, and stakeholder analysis.	Weekly meetings, online research.	Analysis documentation and a clear scope of the project.[2]
2	Do research on existing products (technologies used, the challenges, available open sources)	Online research, brainstorming.	Foresee potential challenges and improve our proposal.
3	Choose hardware (battery, camera, etc.) Budget and resource allocation.	Cost for each component and online information for cheaper alternatives.	Make sure that the required budget is below the $750 threshold.
4	Decide which wearable device to put a camera on; use SolidWorks to lay out the overall structure of the device. 3D print the prototype glasses/headband.	Rough prototypes for each idea to determine which option to use.	Find a more applicable design that would also satisfy user experience.

[1] Some steps can be executed concurrently, and tools used during the process are discussed in Section 3.
[2] Relevant documentation needs to be updated and versioned once there is a change/addition to the team's understanding of the project.

5	Research and construct a solution for a microcontroller module with a camera.	Online research, hardware subgroup discussion.	A microcontroller module for the system.
6	Assemble hardware components.	Pieces of hardware components.	Hardware prototype.
7	Determine the general structure of the software; generate UML and PAC diagrams.	Software subgroup discussions and system analysis.	Software architecture diagrams like UML and PAC.
8	Construct a UI to display the translated text and insert the additional features.	Time and effort.	Make sure that the UI displays information clearly and is user-friendly.
9	Create a proof-of-concept prototype that demonstrates our system's general workflow.	Rough hardware and software components; group meeting.	A draft version that shows this project is feasible.
Proof of concept completed			
10	Implement software solutions to extract texts from images.	Existing machine learning model and domain knowledge.	A software program that extracts texts from images.
11	Program software that would efficiently upload photos to the online platform and receive the translation result.	Time and effort; online resources (existing solutions) if applicable.	Upload the photos as fast as possible to ensure user experience.
12	Establish the connection between hardware and software and test the basic functions.	Available hardware and software; a comprehensive test suite.	A functional product that is thoroughly tested.
Revision 0 completed			
13	Implement possible optimization.	Feedback from professionals and internal reflection on potential product improvements (shape, size, weight, functionalities, etc.).	Make the product more portable, lightweight, functional and reliable.
14	Sustainability considerations for wearable devices.	Existing sustainable technology on wearable devices.	Deliver a viable sustainability plan.
Revision 1 completed			

Table 2: Workflow process

2 Roles and Responsibilities

2.1 Area of Responsibility[3]

Name	Responsibility	General Tasks	Responsible Development Steps
Ding Hao	System integration	Combine mechanical, electrical, and software components.	1, 2, 3, 6, 9, 12, 13, 14
Yimin Pang	Microcontroller	Create a system integrating the microcontroller, camera, and battery.	1, 2, 3, 5, 6, 9, 13, 14
Yucheng Yao	Machine learning model design	Explore existing machine-learning models and integrate appropriate models into the system.	1, 2, 7, 9, 10, 13, 14
Wenyu Yin	Backend programming	Develop backend processes for text extraction and translation.	1, 2, 7, 8, 9, 11, 13, 14
Taoming Yu	Mechanical design	Design the wearable device and position the camera for optimal capture.	1, 2, 3, 4, 6, 9, 13, 14
Xiang Zhang	App development	Develop the user interface for the mobile application.	1, 2, 7, 8, 9, 11, 13, 14

Table 3: Area of responsibility

2.2 Required Standards

- General file naming convention: CamelCase
- Variable naming convention: CamelCase
- Coding and comment format: PEP-8
- Required file type
    - Coding files: .js, .py
    - Image files: .jpg, .png
    - Text files: .txt, .pdf
    - CAD files: .sldprt, .sldasm, .slddrw.
    - AI-translating & Extraction model files: .gguf (depends on the interface)
- Version control standard: Detailed discussed in 2.3
- Security standard: OAuth 2.0 and HTTPS

[3] While each team member has been assigned a specific role, it is crucial that all individuals possess a comprehensive understanding of both current and future tasks related to every aspect of the project. The designated roles serve as a guideline; therefore, collaboration and mutual support among team members are essential to ensure the success of the project as a whole.

2.3 Version Control - Git/GitHub

To maintain code quality, minimize conflicts, and keep track of the entire workflow, the team will follow a clear branching strategy, such as a feature-branch model, to implement version control in GitHub. Each team member will work on separate branches for specific features or tasks. Consistent naming conventions for branches, commits, and pull requests are essential for project standardization.

To achieve effective version control, the team will adopt the following rules:

1. Branching
    - 'main' or 'master': The production-ready branch only contains stable and tested code.
    - 'dev': The develop branch contains the code with completed new features or bug fixes after being tested.
    - Feature branches: Each task or feature should have its own branch, created from the "dev" branch, following the name format 'feature/sub-task.'
2. Commit Messages: Write clear, meaningful, and consistent commit messages which describe any changes. The message should start with verbs like "Added," "Fixed," "Updated," and "Removed."
3. Pull Requests (PRs)
    - Create a pull request before merging any feature branch into the 'dev' branch.
    - PRs should clearly describe what was added, fixed, or changed and any related Issue numbers.
4. Issues
    - Each bug should have a corresponding GitHub Issue, which describes the problem with relevant details.
    - During bug fixing, reference the corresponding Issue in commit messages and PRs to automatically close the Issue after the PR is merged.
    - Each issue should be assigned to the team member responsible for it.
5. Releases
    - A GitHub Release should be created for each significant merge from the 'dev' branch to the 'main' branch, signifying the completion of a set of features.
    - Release versioning should follow the Semantic Versioning convention.
        - MAJOR.MINOR.PATCH (e.g., 1.2.3 represents the 3rd patch in the 2nd minor version within the 1st major version)
    - The team should attach all relevant information to each release, including tags, release notes, related PRs, and built artifacts.
6. Code Review: At least one reviewer should review the changed code before merging. Reviewers should focus on the functionality, coding standards, and potential bugs.
7. Resolve Conflicts: All merged conflicts should be resolved locally and fully tested before pushing any changes to the remote repo.

2.4 Unit Testing

The team should actively follow the TDD (test-driven development) design practice, meaning the test case should be formulated before writing code, programmers should eliminate failing unit tests along the way, and a perfect unit test passing percentage signifies the completion of a feature. In situations where unit tests are not trivial at the beginning of development, team members should always add unit test cases after they finish programming the feature. Developers should ensure that all the unit tests are passed in their local workspace before pushing their local changes to the version control repository.

In general, test cases should satisfy the following attributes:

1. Tests should cover as many edge scenarios as possible.
2. Future changes to the feature should be captured by the unit tests.
3. Unit tests should focus on a specific area (e.g., a variable, a function, etc.) of the code as opposed to integration testing.

Due to the complexity of testing front-ended software programs and the limited resources, UI unit testing is not considered necessary in this project. However, the team should always target a comprehensive unit test coverage for back-end modules and utilities.

2.5 Documentations

When writing documentation for the project, members should adhere to several general requirements to ensure it is practical and useful. General documentation should be consistent in terminology and formatting to maintain clarity and avoid confusion. It should be clear and concise, avoiding unnecessary parts and providing explanations for any technical terms used. The documentation structure should be logical, with well-defined sections such as introductions, objectives, system architecture, user guide, as well as troubleshooting tips. Additionally, each folder should contain a README file to provide context and instructions specific to its content.

It is important to keep track of changes by using version control systems or maintaining a revision history table to document updates. Furthermore, members must consider the target audience and edit the content to meet the needs of both technical team members and end-users.

Software documentation

1. Requirements: Include a detailed list of software requirements, such as operating systems, libraries, and dependencies needed for the application.
2. Installation instructions: Provide step-by-step instructions on how to install and set up the software.
3. User guide: Offer a comprehensive guide on how to use the software, including screenshots or diagrams if necessary.

4. Testing procedures: Describe testing methods used to ensure software functionality and reliability.

Hardware documentation

1. Design specifications: Document the hardware design specifications, including schematics, component lists and measurement details.
2. Assembly instructions: Provide detailed instructions for assembling the hardware components if necessary.
3. Safety information: List any safety precautions or warnings about using hardware components.
4. Troubleshooting: Offers solutions for common issues that may arise during hardware operations.

3 Tools and Technology[4]

Tool	Purpose
SolidWorks	3D modelling software for the mechanical design of wearable devices
Visual Studio Code	Code editing
GitHub	Version control for source code
WeChat/Teams	Communication platform between team members
React Native	Front-end framework for software application
Python	Back-end utility for machine learning-related modules
Arduino	Embedded systems and microcontroller development software
Azure DevOps	Project management and progress tracking
3D Printers	Rapid prototyping and testing of different designs
Google Drive	Documentation collaboration and storage

Table 4: Tools and Technology

[4] Tools and technologies will be updated, and new ones might be introduced while the team delves deeper into the project and does further research.