ENSE810 Semester 2, 2023

Individual Project – Report Template

Student name: Maneesh Maleedu

Student ID: 18033725

Project title: Digital Task Manager

1. User requirements

The user requirements of the Digital Task Manager software include:

1. Ability to add, delete, and mark tasks as complete.

2. Setting task deadlines and receiving notifications for approaching deadlines.

3. Capturing photos and linking them to tasks.

4. Loading and saving tasks from/to files.

5. User-friendly graphical interface for easy interaction.

6. Automatic notifications for tasks with deadlines within 24 hours.

7. Efficient utilization of the Raspberry Pi microcontroller and camera module.

2. Embedded system:

The project can be classified as an embedded system. An embedded system is a combination of hardware and software designed to perform a specific function within a larger system. In this case, the Raspberry Pi microcontroller serves as the hardware platform, and the software developed for task management, photo capture, and notifications constitutes the embedded software. The Raspberry Pi, being a compact and dedicated computing device, is integrated into the system to execute specific tasks, making it an example of an embedded system tailored for digital task management and photo capturing purposes.

3. Requirements analysis

In the development of the Digital Task Manager project, both functional and nonfunctional requirements played a significant role in shaping the application's design and ensuring its effectiveness.

Functional requirements define the specific behaviors and features the software must exhibit to meet user needs. Examples include:

1. Task Management: The application must allow users to add, delete, and mark tasks as completed.
2. Deadline Setting: Users should be able to set deadlines for tasks.
3. Photo Capture: Users should capture and associate photos with tasks.
4. Notifications: The application must notify users when a task's deadline is approaching.

Nonfunctional requirements encompass qualities that enhance the software's performance, usability, and overall experience. Examples include:

1. User Interface Responsiveness: The GUI should respond promptly to user actions, even during camera operations, to maintain a smooth user experience.
2. Notification Accuracy: Notifications should trigger accurately based on task deadlines, ensuring users are informed in a timely manner.
3. Compatibility: The application should run on the Raspberry Pi microcontroller using Python and its associated libraries.

To ensure that the software met these requirements, I conducted systematic testing and validation:

1. Unit Testing: I tested individual components, such as adding tasks, setting deadlines, and capturing photos, to ensure their correctness.
2. Integration Testing: I verified the interaction between different components, like the GUI and camera module, to ensure seamless operation.
3. User Testing: I engaged in user testing to gather feedback on the user interface's intuitiveness and responsiveness.
4. Scenario Testing: I simulated scenarios where tasks' deadlines were approaching to ensure the notification system worked as expected.

Throughout development, continuous monitoring and testing ensured that the application fulfilled both functional and nonfunctional requirements, delivering a robust and user-friendly Digital Task Manager that addressed users' task management needs while maintaining a high-quality user experience.

4. Embedded technologies

In the development of the Digital Task Manager project, I utilized various embedded technologies to bring the application to life, harnessing the power of Linux, Raspberry Pi microcontroller, Python programming, version management, and multithreading. This integration allowed me to create a functional and user-friendly task management application.

Linux, as the operating system running on the Raspberry Pi, formed the foundation of the project. It provided a stable environment for executing the Python code and managing hardware resources. The Raspberry Pi microcontroller itself acted as the hardware platform, allowing me to interface with external devices like the PiCamera and control GPIO pins.

Python, being a versatile and easy-to-learn programming language, played a central role in the project. I leveraged Python to create the user interface using the Tkinter library, design the application's logic, and interact with external hardware. This language choice expedited development and enabled rapid prototyping of the application.

Version management, facilitated by version control systems like Git, allowed for seamless collaboration and code management. Through branches, commits, and merges, I could work on different aspects of the project without worrying about conflicting changes. Additionally, it offered a safety net for experimentation, as I could revert to previous versions if needed.

Multithreading was crucial to ensure smooth user interaction. With the use of Tkinter, the graphical user interface (GUI) ran on the main thread, while a separate thread handled camera functionality. This prevented GUI freezes during camera operations, enhancing user experience. Python's threading module made it straightforward to implement this multithreading approach.

Furthermore, the project's realization highlighted the capabilities of the Raspberry Pi microcontroller. The integration of the PiCamera module, facilitated by the Picamera library, enabled capturing photos directly from the camera module. This hardware integration showcased the potential of embedded systems for diverse applications.

In summary, the project synergized Linux, Raspberry Pi, Python, version management, and multithreading to create a functional Digital Task Manager application. This integration illustrated the versatility of embedded technologies in delivering practical solutions, from graphical user interfaces to hardware interactions. By employing these technologies effectively, I was able to design an efficient and user-friendly application that managed tasks and deadlines while capturing the potential of embedded systems.

5. Testing

The project's testing phase was crucial in ensuring the software's functionality and reliability. The software does indeed accomplish its intended purpose of providing a digital task management solution with the ability to add, delete, mark tasks as complete, set deadlines, capture photos, associate photos with tasks, and receive notifications for approaching deadlines.

The testing approach involved several key methods:

1. Unit Testing: Individual functions and methods were thoroughly tested to ensure they worked as expected. For instance, functions for adding tasks, capturing photos, and setting deadlines were verified in isolation.

2. Integration Testing: The integration of different components, such as the GUI, task management functionality, and camera module, was rigorously tested to ensure proper coordination and communication between these elements.

3. User Testing: The application was tested by end-users to evaluate its user-friendliness and overall experience. User feedback helped identify areas for improvement in terms of usability and interface design.

4. Scenario Testing: Realistic scenarios were simulated to validate critical features. This included adding tasks, setting deadlines, capturing photos, and verifying if notifications were triggered correctly.

The project's testing efforts were successful in validating the software's functionality, identifying, and rectifying issues, and ensuring that the end product met its intended goals. The software effectively addresses task management needs and enhances the user experience through a responsive and intuitive interface.

References

https://projects.raspberrypi.org/en/projects/getting-started-with-picamera

Use of ChatGPT for help with errors in code

https://medium.com/women-make/7-tools-for-gui-development-on-raspberry-pi-67fd7cd9226e

https://linuxhint.com/create-gui-interface-raspberry-pi-using-tkinter/

https://www.techtarget.com/iotagenda/definition/embedded-system

https://machinelearningmastery.com/a-gentle-introduction-to-unit-testing-in-python/