TEAM DELTA  
Project Design

Team Members:

Kalim Dausuel, Marie Thesia Asamba, Nick Geisler, Eric Moore

Summer 2024

**Table of Contents**

[1. Scope 3](#_Toc1480028436)

[Objectives 3](#_Toc476846495)

[Deliverables 3](#_Toc225622675)

[Boundaries: 3](#_Toc1760058207)

[Limitations: 3](#_Toc1808772875)

[2. Requirements 3](#_Toc1400572176)

[Functional Requirements: 3](#_Toc15905882)

[Non-Functional Requirements: 4](#_Toc1871380106)

[3. Methodology 4](#_Toc451576865)

[4. Work Breakdown / Project Tasks 6](#_Toc1633809444)

[Project Setup 6](#_Toc693359494)

[Develop Widget Application (Coding/Testing) 6](#_Toc1577319333)

[Implement Data Management (Phase 2) 7](#_Toc104969267)

[Testing and Quality Assurance (Testing) 8](#_Toc221924394)

[Create Documentation (Documentation) 8](#_Toc1736125434)

[Prepare Final Project Deliverables (Presentation) 9](#_Toc1087082752)

[5. Risks 11](#_Toc1423609586)

[6. Schedule 12](#_Toc2081923567)

[7. Project Evaluation Plan 13](#_Toc2044676853)

[Works Cited 13](#_Toc618875200)

# 1. Scope

Objectives:

* Develop a Screen Time Management App: The primary goal is to create an application that helps users monitor and control their screen time effectively.
* Include Features Like Daily and Session Limits: The app will allow users to set specific time limits for app usage.

Deliverables**:**

A Functional Android App with the Specified Features: The final deliverable will be an Android application that includes alarm and notification settings for time limits.

## Boundaries:

* Focus on Android Devices Only: The project will be developed exclusively for Android devices to streamline development and testing processes.
* Use Existing APIs for Tracking App Usage: Leveraging existing APIs ensures reliable and accurate tracking of app usage, reducing development time and complexity.

## Limitations:

* Limit Analytics to Commonly Used Apps to Ensure Timely and High-Quality Delivery: To manage scope and ensure quality, the app will initially focus on providing analytics for the most commonly used apps.
* Restrict Development to a Manageable Timeframe: The project will adhere to a strict development timeline to ensure completion within the assigned period, focusing on core features first.

# 2. Requirements

## Functional Requirements:

* Alarm Settings: The application should allow the user to set one daily and one session limit measured in hours and minutes. The session timer will reset after the screen has been dark for 15 minutes. The daily timer will reset at midnight every day.
* User Notifications: The application will notify the user when they have reached 100% of either their daily or session limit. The notification will appear over any currently running application.
* User Interface: The application will provide a method to set the daily and session timers immediately after placing the widget on the home screen. The widget will show the time remaining for the daily and session timers.
* Widgets for Quick Access to Main Features: The user will be able to add the application as a widget on their home screen.

## Non-Functional Requirements:

* Performance: The widget will consume less than 15% of the device’s battery over a 24-hour period. The widget will not consume more than 50 MB of RAM during regular operation. The size of the APK file will not exceed 10 MB. The widget will respond to user interaction within 2 seconds. The method to set or reset the daily and session timers will allow users to complete the process within 1 minute for each timer.
* Privacy: User data will not be exposed to unauthorized parties and the data of the users will be processed in accordance with established privacy policies.
* Compatibility: The application will be compatible with Android devices running Android 12.0 and above. The application will pass compatibility tests on at least two different emulated devices and two real world devices across various manufacturers and OS versions.
* Usability: The application will be designed in a way that is simple to navigate and intuitive. The application should not be complicated in a way that will only be understood by a professional programmer alone.

# 3. Methodology

We have determined that despite Agile's popularity in project management, a Waterfall approach is more suitable for our project’s academic setting. As noted by Radigan (n.d.), Waterfall follows "a clearly defined sequence of execution with project phases that do not advance until a phase receives final approval." This sequential process is particularly beneficial for projects with predictable processes and well-defined requirements, which is the current situation in which project guidelines with hard deadlines are clearly outlined from the start.

One of the key strengths of Waterfall for our project is its emphasis on comprehensive planning and documentation. Radigan (n.d.) points out that Waterfall provides "better focus on documentation of designs and requirements" and ensures "the design phase is more methodical and structured before any software is written." Again, this aligns well with an academic setting, where thorough documentation is a crucial component of project evaluation. Additionally, the Waterfall model's rigorous documentation practices will benefit us since our team is geographically dispersed. The documentation that we collaboratively develop can provide any needed answers in lieu of real time communication.

While Agile methodologies excel at accommodating changes on the fly, our project's short timeline, geographic distance between team members, and hard deadlines make such flexibility a potential source of unnecessary chaos. In contrast, the Waterfall model provides an agreed-upon plan that can be executed asynchronously, which is particularly advantageous for our team's distributed nature. This asynchronous execution allows team members to work independently on their assigned tasks within each phase, reducing the need for constant real-time communication and coordination (Stoica et al., 2013). Furthermore, the clear, linear progression of Waterfall phases makes it easier to communicate project status and progress to stakeholders, including our professor and peers (Petersen et al., 2009).

The Waterfall model's clear project phases can also help in managing the fixed timeline and specific deliverable deadlines of our academic project. As Radigan (n.d.) states, "The cost of the project can be estimated after the requirements are defined." This translates to better time management and resource allocation, allowing us to plan our work around key submission dates for our design document, test plan, source code, and final presentation. The structured nature of Waterfall can help ensure that we meet these critical deadlines, which is essential for successful project completion and grading. Moreover, the Waterfall approach's emphasis on upfront planning and design can lead to fewer errors and reduced development time, which is particularly beneficial given our project's tight academic schedule (Balaji & Murugaiyan, 2012).

Another significant advantage of the Waterfall model for our project is its ability to mitigate risks associated with team member turnover or unavailability. In an academic setting, where team members might have varying schedules or unforeseen commitments, the detailed documentation and clear phase structure of Waterfall ensure that the project can progress even if a team member is temporarily unavailable or needs to be replaced. This contrasts with Agile methodologies, which often rely heavily on tacit knowledge and frequent team interactions, making them more vulnerable to disruptions in team composition (Ahimbisibwe et al., 2015). The comprehensive documentation in Waterfall serves as a knowledge repository, allowing new team members to quickly understand the project's status and requirements, thus minimizing potential setbacks.

Furthermore, the Waterfall approach aligns well with the evaluation criteria typically used in academic settings. Unlike Agile, which focuses on working software as the primary measure of progress, Waterfall produces tangible artifacts at each phase that can be easily evaluated and graded. This includes detailed requirements documents, design specifications, test plans, and final deliverables. These artifacts not only serve as checkpoints for our team but also provide our professor with clear benchmarks for assessing our progress and understanding of the course material (Papadopoulos, 2015).

# 4. Work Breakdown / Project Tasks

## Project Setup

1.1 Set Up Communication Channel (1-2 days) - (Planning/Design)

* **Personnel:** Kalim (lead, 8-10 hours), Marie (support, 4-6 hours)
* **Resources:** Microsoft Teams (school account), personal computers
* **Cost:** $0 (using school-provided software)

1.2 Set Up Dev Environment (10-14 days) - (Planning/Design)

* **Personnel:** Nick (lead, 40-50 hours), Eric (support, 20-25 hours), Marie (support, 20-25 hours)
* **Resources:** Personal computers, Android Studio, SDK tools, school's technical library
* **Cost:** $0 (free development tools and educational resources)

1.3 Set Up GitHub Repository (5-7 days) - (Planning/Design)

* **Personnel:** Eric (lead, 20-25 hours), Nick (support, 10-15 hours)
* **Resources:** GitHub Education account, Git software
* **Cost:** $0 (free for students)

## Develop Widget Application (Coding/Testing)

2.1 Research Widget Development (13-15 days) - (Phase 1)

* **Personnel:** All team members (20-25 hours each)
* **Resources:** School's technical library, online tutorials, Android documentation
* **Cost:** $0 (using educational and free online resources)

2.2 Develop Proof of Concept (7-9 days) - (Phase 1)

* **Personnel:** Marie (lead, 30-35 hours), Nick (support, 15-20 hours), Eric (support, 15-20 hours)
* **Resources:** Personal computers, Android Studio, Android emulator
* **Cost:** $0 (using free development tools)

2.3 Create Basic Widget Structure (2-4 days) - (Phase 1)

* **Personnel:** Kalim (lead, 16-20 hours), Marie (support, 8-10 hours)
* **Resources:** Personal computers, Android Studio, school's technical library
* **Cost:** $0 (using existing equipment and educational resources)

2.4 Implement Screen Time Tracking (6-8 days) - (Phase 2)

* **Personnel:** Nick (lead, 25-30 hours), Eric (support, 15-20 hours)
* **Resources:** Personal computers, Android Studio, Android emulator
* **Cost:** $0 (using free development tools)

2.5 Develop Usage Limit Features (6-8 days) - (Phase 2)

* **Personnel:** Eric (lead, 25-30 hours), Kalim (support, 15-20 hours)
* **Resources:** Personal computers, Android Studio, Android emulator
* **Cost:** $0 (using free development tools)

2.6 Create Notification System (6-8 days) - (Phase 2)

* **Personnel:** Marie (lead, 25-30 hours), Nick (support, 15-20 hours)
* **Resources:** Personal computers, Android Studio, Android emulator
* **Cost:** $0 (using free development tools)

## Implement Data Management (Phase 2)

3.1 Research Data Storage Options (2-3 days) - (Phase 1)

* **Personnel:** Kalim (lead, 12-15 hours), Eric (support, 6-8 hours)
* **Resources:** School's technical library, online resources
* **Cost:** $0 (using educational and free online resources)

3.2 Set Up Local Data Storage (4-6 days) - (Phase 1)

* **Personnel:** Eric (lead, 20-25 hours), Marie (support, 10-15 hours)
* **Resources:** Personal computers, Android Studio, SQLite (built-in)
* **Cost:** $0 (using free development tools)

3.3 Optimize Data Operations (2-4 days) - (Phase 2)

* **Personnel:** Nick (lead, 16-20 hours), Kalim (support, 8-10 hours)
* **Resources:** Personal computers, Android Studio, Android profiling tools
* **Cost:** $0 (using free development tools)

## Testing and Quality Assurance (Testing)

4.1 Research Testing Methodologies (2-3 days) - (Phase 1)

* **Personnel:** Marie (lead, 12-15 hours), Kalim (support, 6-8 hours)
* **Resources:** School's technical library, online testing resources
* **Cost:** $0 (using educational and free online resources)

4.2 Perform Unit Testing (9-11 days) - (Testing)

* **Personnel:** Nick (lead, 35-40 hours), Eric (support, 20-25 hours)
* **Resources:** Personal computers, Android Studio, JUnit, Mockito
* **Cost:** $0 (using free testing tools)

4.3 Conduct Integration Testing (9-11 days) - (Testing)

* **Personnel:** Kalim (lead, 35-40 hours), Marie (support, 20-25 hours)
* **Resources:** Personal computers, Android Studio, Espresso, Android emulator
* **Cost:** $0 (using free testing tools)

4.4 Perform User Acceptance Testing (2-3 days) - (Testing)

* **Personnel:** All team members (10-12 hours each)
* **Resources:** Personal computers, Android emulator, team members' personal devices
* **Cost:** $0 (using existing devices)

## Create Documentation (Documentation)

5.1 Research Documentation Best Practices (1-2 days) - (User Guide)

* **Personnel:** Eric (lead, 8-10 hours), Nick (support, 4-6 hours)
* **Resources:** School's technical library, online resources
* **Cost:** $0 (using educational and free online resources)

5.2 Develop User Guide (14-16 days) - (User Guide)

* **Personnel:** Marie (lead, 50-60 hours), Kalim (support, 25-30 hours)
* **Resources:** Personal computers, Google Docs, free screen capture tools
* **Cost:** $0 (using free tools)

5.3 Prepare Technical Documentation (14-16 days) - (Presentation)

* **Personnel:** Kalim (lead, 50-60 hours), Eric (support, 25-30 hours)
* **Resources:** Personal computers, Google Docs, free diagramming tools
* **Cost:** $0 (using free tools)

## Prepare Final Project Deliverables (Presentation)

6.1 Finalize Source Code (2-3 days) - (User Guide/Presentation)

* **Personnel:** Nick (lead, 12-15 hours), Eric (support, 6-8 hours), Marie (support, 6-8 hours)
* **Resources:** Personal computers, Android Studio, GitHub
* **Cost:** $0 (using free development tools)

6.2 Create Presentation Materials (13-15 days) - (Presentation)

* **Personnel:** Kalim (lead, 40-50 hours), Marie (support, 20-25 hours)
* **Resources:** Personal computers, Google Slides, free design tools
* **Cost:** $0 (using free presentation tools)

6.3 Finalize GitHub Repository (13-15 days) - (Presentation)

* **Personnel:** Eric (lead, 40-50 hours), Nick (support, 20-25 hours)
* **Resources:** Personal computers, GitHub, documentation tools
* **Cost:** $0 (using free tools)

# 5. Risks

This section highlights the risks involved with programming an Android application in such a short period with a small team. Recognizing these risks is important when planning for the development of the app and will ensure that we are prepared to put the right amount of effort into avoiding any potential pitfalls the team may encounter.

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Probability** | **Impact** | **Action** |
| Unfamiliarity with Android Studio | 70% | Not having used this environment can slow down the development of the application as it includes many different features that the programmers and UI Designers will need to be familiar with. | Spend time playing around with the application and make sure that everyone has configured their environment in a similar way to prevent confusion. |
| Unfamiliarity with Application Development | 90% | Not having developed applications before means that team members can be more likely to make common mistakes that more experienced developers might avoid | Program one feature at a time to make sure the feature is functional before moving on to new tasks and use online resources such as Stack Overflow to see what other developers have done in the past to solve similar problems. |
| Team Availability | 100% | Members live in different time zones and therefore communication might be slow or strained. | Each team member has agreed to use MS Teams and check in at least once per day to update on what is being worked on and any challenges impacting the project. |

# 6. Schedule

Below is a chart highlighting the deadlines and expected time frame required to create the application.

|  |  |  |  |
| --- | --- | --- | --- |
| Application Development Timeline |  |  |  |
|  |  |  |  |
| Tasks | Start Date | Duration (Days) | End Date |
| 1. Planning/Design | 06/18/2024 | 14 | 07/02/2024 |
| 1.1 Project Plan | 06/18/2024 | 7 | 06/25/2024 |
| 1.2 Project Design | 06/25/2024 | 7 | 07/02/2024 |
| 2. Coding/Testing | 07/02/2024 | 21 | 07/23/2024 |
| 2.1 Phase 1 | 07/02/2024 | 7 | 07/09/2024 |
| 2.2 Testing | 07/09/2024 | 7 | 07/16/2024 |
| 2.3 Phase 2 | 07/16/2024 | 7 | 07/23/2024 |
| 3. Documentation | 07/23/2024 | 14 | 08/06/2024 |
| 3.1 User Guide | 07/23/2024 | 7 | 07/30/2024 |
| 3.2 Presentation | 07/30/2024 | 7 | 08/06/2024 |

# 7. Project Evaluation Plan

This section defines how Team Delta plans to evaluate the team’s progress during the development of the application. The evaluation can be simplified into four major criteria, the functionality of the application, the development of the UI, how the program interacts with the UI, and the impact the application has on device performance. The table below shows Team Delta’s goals during the two phases of programming and the testing phase. A phase can be evaluated as complete if the goals below are achieved within the allotted timeframe defined in the schedule found in section 6 of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Phase 1 | Testing Phase | Phase 2 |
| Programming the app’s functionality | Code the app to run two timers simultaneously and only run while the phone is not asleep. | Look for flaws in the application, to be sure that the code can be adjusted as necessary in phase 2. | Adjust or improve any code that is clunky or does not work as intended. |
| Creating the UI that the app will use | Create a simple UI that allows a user to set both a session timer and daily timer for phone usage. | Check that the UI is smooth and works as intended. | Change any parts of the UI that seem unintuitive or are difficult to interact with |
| Combining the UI with the app’s intended functionality | Combine the UI with the app’ programming to create a full product. | Make sure that interactions within the UI prompt the expected results from the application’s programming | Fix any connections between the UI and the programming if they do not seem to be interacting properly during the testing phase. |
| Device performance while running the application | The device should be able to run other apps while the timer is active, so it is important to program the application to be as lightweight as possible. | Check that the device’s performance is not impacted by the application by using the timer while actively using other apps on the phone. | Try to reduce stress on the user’s device if the application seemed to impact performance while the app is in use. |

# Works Cited

Ahimbisibwe, A., Cavana, R. Y., & Daellenbach, U. (2015). A contingency fit model of critical success factors for software development projects: A comparison of agile and traditional plan-based methodologies. Journal of Enterprise Information Management, 28(1), 7-33.

Balaji, S., & Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. International Journal of Information Technology and Business Management, 2(1), 26-30.

Palmquist, S., Lapham, M. A., Miller, S., Chick, T., & Ozkaya, I. (2013). Parallel worlds: Agile and waterfall differences and similarities. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.

Papadopoulos, G. (2015). Moving from traditional to agile software development methodologies also on large, distributed projects. Procedia-Social and Behavioral Sciences, 175, 455-463.

Petersen, K., Wohlin, C., & Baca, D. (2009). The waterfall model in large-scale development. In International Conference on Product-Focused Software Process Improvement (pp. 386-400). Springer, Berlin, Heidelberg.

Radigan, D. (n.d.). Agile vs. waterfall project management. Atlassian.<https://www.atlassian.com/agile/project-management/project-management-intro>

Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software Development: Agile vs. Traditional. Informatica Economica, 17(4), 64-76.