TeamLead Application

**Project Report**

**Table of Contents**

[Introduction 4](#_Toc482090617)

[About this Document 4](#_Toc482090618)

[Scope 4](#_Toc482090619)

[Purpose 4](#_Toc482090620)

[References 4](#_Toc482090621)

[Software Requirements 5](#_Toc482090622)

[Conventions 5](#_Toc482090623)

[Functional Requirements 5](#_Toc482090624)

[Non-Functional Requirements 6](#_Toc482090625)

[Software Design 7](#_Toc482090626)

[Overview 7](#_Toc482090627)

[User Interface 7](#_Toc482090628)

[Task Grid Subsystem 9](#_Toc482090629)

[Workday Data Model 10](#_Toc482090630)

[Charting and Visualization 12](#_Toc482090631)

[Testing 14](#_Toc482090632)

[Future work 16](#_Toc482090633)

**Version History**

The following table depicts the revision history of this document.

| Author | Description | Date | Sections |
| --- | --- | --- | --- |
| James Williamson | Original draft | 05/07/2017 | All |
|  |  |  |  |

# Introduction

## About this Document

This document describes the construction of the TeamLead application for EEC 626 – Software Engineering Project, Spring 2017 section. It contains information on the development effort, requirements, design information, and information on future work.

## Scope

This document covers the design and implementation of the TeamLead application for Andorid.

## Purpose

Leading a software development team brings with it several unique challenges. A software team lead must be adept at development, coaching/mentoring, architecture, release engineering, project planning, and more. This requires the ability to switch tasks and contexts fluidly, and demands superior time management skills.

As such, the TeamLead application is proposed in order to aid software team leads by empowering them to leverage of one of today’s most ubiquitous tools – the smartphone. There are myriad software applications currently available for time management, but most are either too generic or heavyweight to be consistently useful for the unique demands of this discipline. Worse, the learning curve may be too steep; this is an important consideration in a field where countless tools, frameworks, languages, and platforms must be adopted and utilized. TeamLead seeks to provide a streamlined, user-friendly, and intuitive way for team leads to solve this problem.

## References

[1] TeamLeadApplication User Manual (<https://github.com/JWilliamson45/TeamLead/tree/master/docs>)

# Software Requirements

This section lists the software requirements specification for the TeamLead application.

## Conventions

The requirements listed in this section use different verbs to indicate necessity:

* “Shall” – The requirement must be implemented for proper operation of the system.
* “Should” – The requirement may be implemented, subject to time and complexity constraints.

Nested requirements may be listed under base requirements, in some cases. These should be treated as separate individual requirements; this practice is intended to help convey context. These are distinguished by indentation and sub-numbering throughout the document.

x.y) Base requirement

x.y.z) Nested requirement

## Functional Requirements

1.1) The application shall allow the user to track time spent on various tasks throughout their workday via simple “task” button presses.

1.1.1) When the user clicks a task button, the application shall log a context switch and begin recording time spent on the specified task.

1.2) The application shall allow the user to add custom task buttons.

1.2.1) Each task shall be identified by a name string, which is displayed on the top of the button.

1.2.2) Each task shall maintain a timer depicting the total time elapsed for the task, which is displayed on the bottom of the button.

1.2.3) The application should support the capability for the user to choose a custom color for each task button.

1.3) The application shall allow the user to delete specified task buttons.

1.4) After a button is pressed, the application shall lighten the button color, to indicate that it is the active task.

1.5) After a button is pressed, the application shall repaint the ContextSwitch activity UI every 100 milliseconds to show timer feedback to the user.

1.5.1) The application should support the capability for the user to adjust the repaint interval to values of 100mS, 500mS, 1 second, or 5 seconds.

1.6) The application shall allow the user to specify a “threshold” time for a task, which is the maximum desired amount of time to spend daily on that activity.

1.6.1) When the configured threshold time for a task is exceeded, the application shall support the capability to generate a visual alarm.

1.6.2) When the configured threshold time for a task is exceeded, the application should support the capability to generate an audible alarm.

1.7) The application shall generate a “task log” that shows the user a list of each performed iteration and the total time spent on each.

1.8) Once the workday is concluded, the application shall generate a graph to illustrate to the user how their time was allocated throughout the day.

1.8.1) The application shall support the capability to generate the workday graph in pie chart format.

1.8.2) The application should support the capability to generate the workday graph in time-slice format.

1.8.3) The application should allow the user to index up to ten previous recorded workdays for analysis.

1.8.4) The application should display the total number of context switches to the user.

1.9) The application shall support the ability to be displayed on the lock screen in focus, facilitating ease of data entry and usability without necessitating the need to unlock a user’s phone.

## Non-Functional Requirements

2.1) The application is supported for only the Android operating system.

2.1.1) The minimum API (Android SDK) compatibility requirement is API level 19: Android 4.4 (“Kitkat”).

2.1.2) The target API (Android SDK) compatibility requirement is API level 25: Android 7.1 (“Nougat”).

2.2) The application requires permission to write to external storage on the device.

# Software Design

## Overview

The TeamLead application is written in Java for the Android operating system. It utilizes Java and Android APIs and standard techniques for Android application construction. The application is written for mid- and large-format mobile phones; it is not optimized for tablets and does not support Android Wear or Android TV systems.

## User Interface

The application user experience is constructed around five primary “activities,” which are Android window objects and essential application building blocks that facilitate user interaction. Figure 1 depicts these activities; each interact with the Workday – which acts as the data model – via reference to the TeamLeadApplication.



Figure 1: Application User Interface

The **WorkdaySummaryActivity** is used to present the output of the Workday to the user in the form of a graph.

The **ContextSwitchActivity** is the main display. This activity displays a grid of task tiles that correspond to things a typical user spends their workday doing. Each time a new task tile is clicked, the system records the change and updates a ticking timer depicting the amount of total time spent on the task. It can be shown directly from the device lock screen, if applicable, when in focus. For more detail on how to use the UI, refer to [1].

The **SettingsActivity** is the screen for handling application preference adjustments.

The **TaskLogActivity** is an informative display that can show the complete task log, with each iteration in sequence, to the user. Times are shown next to each iteration.

The **AddNewTaskActivity** is used to create additional tasks and set task options.

In addition to these activities, there are various dialogs and fragments that may be used to show context menus, informative snippets, and errors.

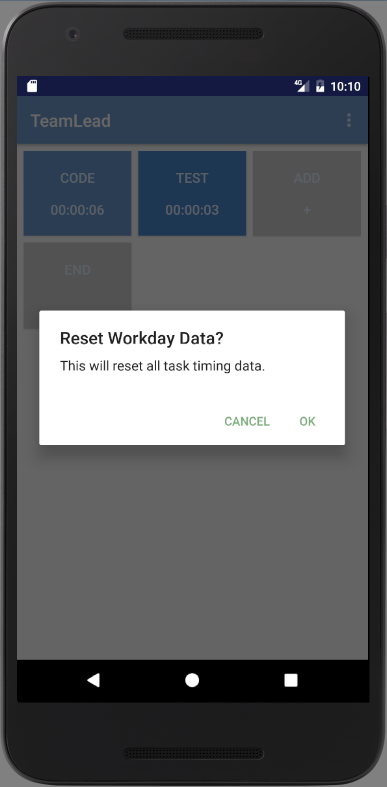


Figure 2: Example Dialog Fragment in Focus

The application uses a consistent theme and toolbar structure for a uniform look and feel.

## Task Grid Subsystem

The task grid subsystem used by the ContextSwitchActivity is shown in Figure 3. **Green** boxes depict Android/Java objects, while **blue** boxes depict application code.

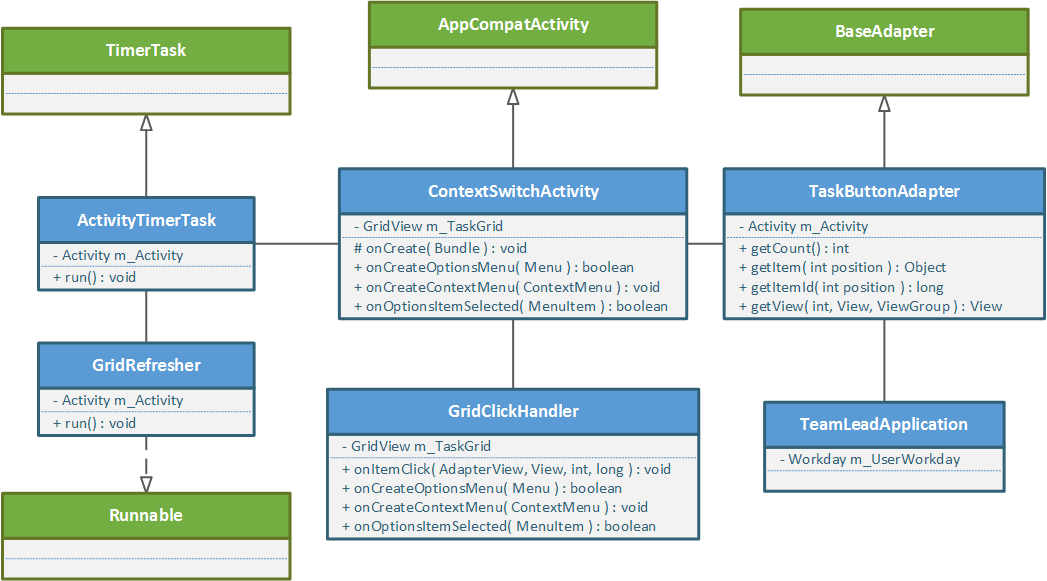


Figure 3: Task Grid Subsystem UML Diagram

The ContextSwitchActivity uses a custom adapter that inflates (renders objects from XML) task buttons to be displayed on the GridView that the user interacts with. This adapter – TaskButtonAdapter – subclasses the BaseAdapter class. It can paint buttons (also referred to as task tiles) differently using information from the Workday data model, which is accessed via TeamLeadApplication.

When a user clicks the grid, the GridClickHandler registers the input and determines which button, if any, was touched. Any user-defined task tile is eligible; the “special” persistent task tiles marked “add” and “end” perform specific actions that launch other activities.

The ActivityTimerTask is used to force a repaint of the UI through periodic invalidation of the GridView. It schedules a GridRefresher “Runnable” class at the specified interval. This is used to achieve the “ticking timer” effect when tasks are active and the UI is in focus.

## Workday Data Model

The Workday object shown in Figure 4 serves as the data model for the TeamLead application; it stores all details about the active workday.

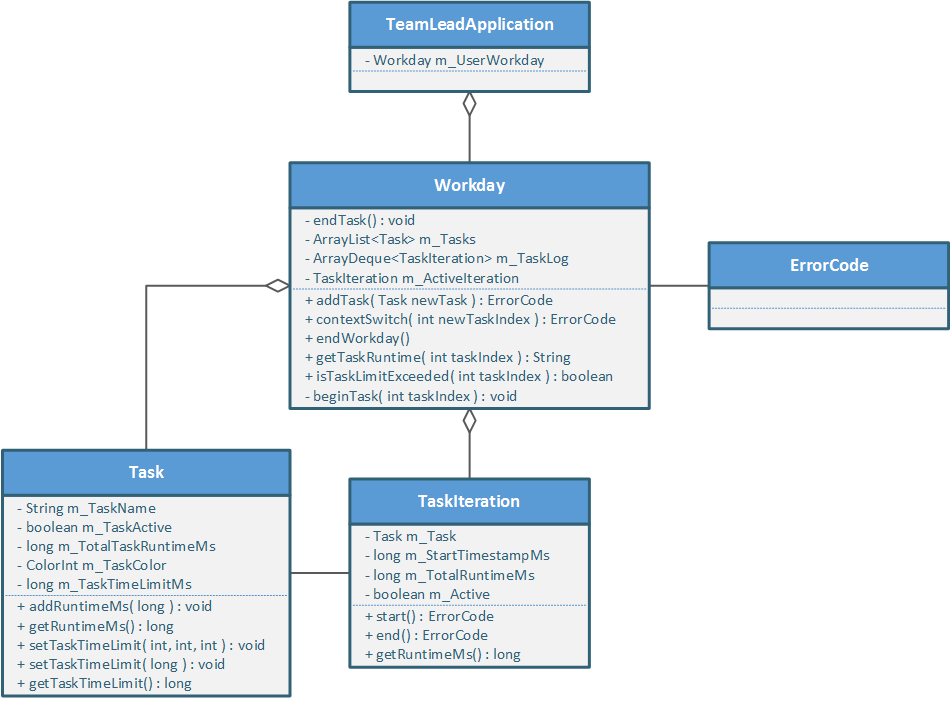


Figure 4: Workday Model UML Diagram

The Workday predominantly models two things: the list of user-defined tasks that have been created and are displayed as task tiles on the UI (an ArrayList of Task objects), and an ordered list of each task iteration performed, since tasks may be performed more than once and not necessarily in a contiguous fashion (an ArrayDeque of TaskIteration objects).

A Task encapsulates all of the details about a user-defined task, including the name (which serves as the unique identifier), a flag indicating whether or not the task is currently being performed, total task runtime in milliseconds, an associated task color, and an optional task time limit in milliseconds. Each Task corresponds to a button on the ContextSwitchActivity.

A TaskIteration models a single run on a given Task. As such, it must have an associated Task type. It uses a start timestamp and end timestamp against the device’s system runtime clock to determine total execution time. Pressing a task tile on the ContextSwitchActivity creates a new TaskIteration and records a starting timestamp. Once any other user-defined task tile is pressed, the TaskIteration marks itself as inactive, records an end timestamp, and is added to the ArrayDeque in the Workday. In this manner, it can be recalled later as part of the complete set for display on the TaskLog (or in order to be written to a file).

Error codes are used where appropriate to convey that an operation has gone wrong.

The TeamLeadApplication (which subclasses the Android Application class) keeps a global instance of the Workday for retrieval by the various Android Activities. If the application is prematurely shut down, whether by the OS or the user, before the Workday is completed, the Workday object is serialized and written to a file for data persistence. Then, on subsequent invocation, the Workday is read from the file and started again.

## Charting and Visualization

When a Workday is concluded, the user must be able to obtain concise visual feedback of how their time was allocated. The application currently supports the generation of a custom PieChart widget that is illustrated in Figure 5.

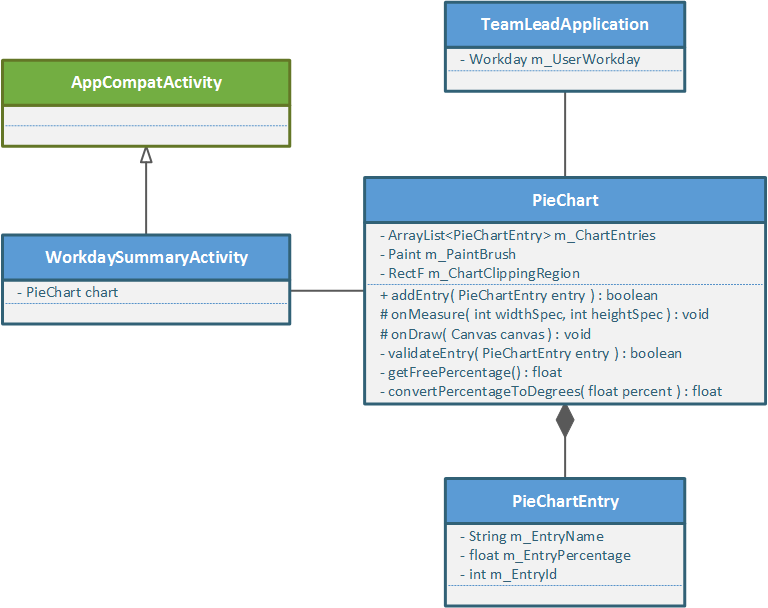


Figure 5: Pie Chart View UML Diagram

The PieChart extends the Android View class and is used to measure and render a custom widget on the UI. It maintains a list of PieChartEntry objects, which model the actual segments of the chart based on assigned percentages and a given label. Its onDraw method paints each segment to produce a full chart.

**Pseudocode listing for PieChart onDraw() method:**

* Initialize “percentage tally” variable to zero.
* Specify clipping region around top-center of canvas.
* Iterate through all PieChartEntry objects:
  + Retrieve next color from list (up to 20 currently) for paintbrush
  + Set segment “start angle” to percentage tally converted to degrees
  + Set segment “sweep angle” to PieChartEntry percentage converted to degrees
  + Draw segment using drawArc()
  + Draw corresponding label underneath the chart area
  + Update percentage tally

**Pseudocode listing for PieChart onMeasure() method:**

* If parent leaves measurement spec unspecified, use native size.
* Else, take dimensions that are offered and scale appropriately.

The WorkdaySummaryActivity creates a new PieChart when the Workday is concluded.

# Testing

No integrated unit testing was created and no formally structured integration or functional testing occurred due to project manpower and time constraints.

Basic developer testing was performed on each demo/delivery milestone (versions 0.1.0 through 0.6.0) to ensure basic cases were covered:

1. Can the user add a new task?
   1. Will a duplicate task name be rejected?
   2. Will an invalid task string be rejected?
   3. Will an excessively long task string be rejected?
2. Can the user delete a task?
   1. Can the user delete only user-defined tasks?
3. Can the user select a task and watch the timer update?
   1. Does only the active task update?
   2. Is time accounted for properly?
4. Can the user edit an active task?
5. Can the user assign a custom task color?
   1. Does the “color preview” update appropriately?
6. Can the user assign a custom task alarm threshold?
   1. Does it activate properly?
   2. Does no threshold (zero value) work properly?
7. Can the user reset the Workday data?
8. Can the user change the repaint interval for the ContextSwitch UI?
   1. 100ms work properly?
   2. 500ms work properly?
   3. 1s work properly?
   4. 5s work properly?
9. Is the TaskLog accurate following several context switches?
   1. Is it accurate following a Workday reset?
   2. Is it accurate following several more new context switches following a reset?
10. Does the PieChart render properly for a full Workday?
    1. Is the data (segments) accurate when compared to the TaskLog?
11. Does the PieChart render properly for a blank Workday?
12. Does the app successfully save Workday state upon a “force close” from Android?

Construction and execution of a formal test plan is planned in the “Future Work” section.

# Future work

TeamLead is open-source and publicly hosted (<https://github.com/JWilliamson45/TeamLead/>), and the goal is to continue development beyond the end of the course and continue to improve the application through the addition of new features and bug fixes.

The application at the time of final demonstration is versioned at 0.6.0, and requires some work to get it to initial release candidacy (v1.0.0).

In this section is a list of future work to consider to finish implementation and provide further enhancement to functionality:

* Store Workday data to files in XML format, rather than binary format. This would improve compatibility and human-readability of the stored data.
* Integrate with the cloud to store data or make it available on other mediums (e.g. desktop).
* Add audible alarms when a task time limit is exceeded.
* Add different output types (in addition to pie chart format).
* Add support for small and extra-large form factor devices.
* Add smartwatch integration support.
* Create formal test plan and automated tests.
* Expand preferences menu to account for:
  + Look and feel (theme) adjustments
  + Text and/or grid size
  + Task Log display modifications
  + WorkdaySummary display modifications
  + File output type
  + Cloud integration settings