ACM Word Template for SIG Site

Alan Koruth  
801 Coker Dr  
919-909-6839

koruth@live.unc.edu

Franz Dominno  
700 Bolinwood Drive, Apt 40A  
Chapel Hill, NC 27514  
919-225-7172

dominno@live.unc.edu

Victor Cui

University of North Carolina at Chapel-Hill

4257 Laurel Ridge Circle

Weston FL, 33331

+1 (954) 646-9764

victorvi@live.unc.edu

**ABSTRACT**

In this paper, we describe the formatting guidelines for ACM SIG Proceedings.

**CCS Concepts**

• **SQLite Database**   • **Using Separate threads in timers ➝Massively parallel and high-performance simulations.**This is just an example, please use the correct category and subject descriptors for your submission*.* The ACM Computing Classification Scheme:

<http://www.acm.org/about/class/class/2012>. Please read the [HOW TO CLASSIFY WORKS USING ACM'S COMPUTING CLASSIFICATION SYSTEM](http://www.acm.org/publications/article-templates/CCS-HOWTO-v6-12Jan2015.docx) for instructions on how to classify your document using the 2012 ACM Computing Classification System and insert the index terms into your Microsoft Word source file.

**Keywords**

Keywords are your own designated keywords separated by semicolons (“;”).

# INTRODUCTION

# Studying is one of the most important activities college students do, and one they spend a lot of time on. It’s recommended that for each hour students spend in class, they should spend approximately 2-3 hours studying for that class. With the average student carrying a course load of 15 credit hours, this equates to 30-45 hours of studying per week. Unfortunately with the rise of the internet and social media, students are becoming ever more distracted, taking away from their study time. In one study, researchers found that at a northeastern university, female first-year college students spend nearly 12 hours a day using social media such as Facebook or Twitter, which may correlate to lower GPA’s [1]. Smartphones are one of the biggest mediums used by college students to access social media, but they can also help a student manage their time and focus better. Therefore, the authors decided to create an app that helps students maximize their study time and time management skills. We created an android app with three tabs, “Todo”, “Calendar”, and “Timer”. The Timer tab is based on a time management method designed by Francesco Cirillo, and it emphasizes taking frequent breaks to improve mental agility.

# 2. PAGE SIZE

All material on each page should fit within a rectangle of 18 × 23.5 cm (7" × 9.25"), centered on the page, beginning 1.9 cm (0.75") from the top of the page and ending with 2.54 cm (1") from the bottom. The right and left margins should be 1.9 cm (.75"). The text should be in two 8.45 cm (3.33") columns with a .83 cm (.33") gutter.

# TYPESET TEXT

## Normal or Body Text

Please use a 9-point Times Roman font, or other Roman font with serifs, as close as possible in appearance to Times Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

## Title and Authors

The title (Helvetica 18-point bold), authors' names (Helvetica 12-point) and affiliations (Helvetica 10-point) run across the full width of the page – one column wide. We also recommend phone number (Helvetica 10-point) and e-mail address (Helvetica 12-point). See the top of this page for three addresses. If only one address is needed, center all address text. For two addresses, use two centered tabs, and so on. For more than three authors, you may have to improvise.[[1]](#footnote-1)

## First Page Copyright Notice

Please leave 3.81 cm (1.5") of blank text box at the bottom of the left column of the first page for the copyright notice.

## Subsequent Pages

For pages other than the first page, start at the top of the page, and continue in double-column format. The two columns on the last page should be as close to equal length as possible.

Table . Table captions should be placed above the table

|  |  |  |  |
| --- | --- | --- | --- |
| **Graphics** | **Top** | **In-between** | **Bottom** |
| Tables | End | Last | First |
| Figures | Good | Similar | Very well |

## References and Citations

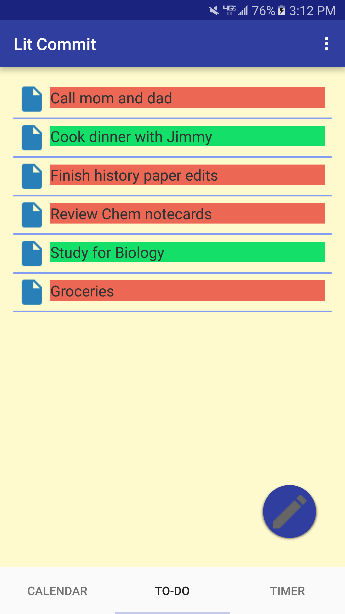
Footnotes should be Times New Roman 9-point, and justified to the full width of the column.

# To-do Tab

The to-do tab of the Lit Commit app provides another use for those looking to maximize their productivity whilst studying. Organization is a crucial value to have as a student. If you look some of the leaders in the business world, one thing you will find in common is that their organizational skills are top notch. More than likely, they will have a calendar, detailed to the day, outlining the tasks for the day. This way, time is not lost figuring out what needs to be done or where to allocate resources. Now, scale that down to the needs of a college student. A typical one will have class, assignments, papers, club meetings, team practices, social activities, and many more to balance. It can all be very stressful for an 18 year old to manage all this. The best way to manage this stress, is to organize oneself; list out the things that need to be done and when. Since one day varies so much from the next for a student and commitments come up unexpectedly, a calendar can be of less use. Instead creating a to-do list the night before a busy day becomes useful. This is where the to-do tab of this app comes into play.

This tab is designed like a to-do list. A user can add a to-do item to the list by pressing the add button in the lower right corner. This brings up the editor mode. The user can then type whatever it is they need and as much as they need. Pressing the back button then saves the item. As more and more items are added the main activity turns into a scrollable list. All items in the list are sorted in chronological order with the oldest items at the bottom. The main activity only shows the first line of each to-do item. This way the user can go into more explanation of what they want in the actual note itself. Pressing on an existing note takes the user to the editor mode once again. They can update the note from here, but cannot delete the note in editor mode. This way if the user accidentally deletes all text and presses back, his to-do item is not gone. Additionally the delete options in the main activity encourage the user to finish an item on the to-do list before throwing it out.

Another feature implemented is the marked read or not characteristic, similar to email. When updating an existing note, and the user should do this only while updating, the user can press a check mark on the bottom of the screen. If the check mark is green, it marks the item finished. If red, the item is still to be finished. This translates to the main activity that all unfinished items are highlighted red while all finished items are highlighted green.

**Figure 2.1** To-Do tab main

Delete options are available in the menu of the main activity toolbar. The options are “delete all checked” and “delete list”. The “delete all items” is self-explanatory but the other option deletes all to-do items that have been marked finished, or highlighted green.

## Implementation

The To-do items were tracked using a SQLite database. Android has an integrated SQLite implementation that stores the database internally which made the implementation easier in the sense that an external connection to a database was not needed. Although this did make debugging database operations difficult as the database could not be visually seen. The schema of the database was organized as follows: a field for item id, which was auto incremented, a field for item text, a field for date created using SQLite’s timestamp, and a field for whether the item had been marked read or not.

This entire portion of the project was handled with traditional structure of android app implementing backends. A custom version of SQLite’s OpenHelper was used to create the database schema and the database itself. Then, a content provider was used to provide access to the physical database. The custom content provider used in the to-do tab encapsulated all necessary database operations so that other running code could delegate these operations. All data passed to the content provider or read calls were organized with URI objects. Uri’s were structured so that the default path to the database was stored while the last segment of the path was the parameter with which a record could be identified.

All read/write calls to the database were made from the main to-do activity and the secondary editor activity. A custom implementation of the cursor adapter was used to append all to-do items to the list view. This implementation included the logic to update text when a new note is pressed and to change if the current item, retrieved by a cursor, had been finished or not.

Android apps that use backends could potentially have databases with large amounts of data. It is not wise for these apps to implement database operations on the main thread as it could force the app to crash. With the scale of lit commit, this issue doesn’t occur as data is small but to comply with good practice all database operations were moved from the main thread. The main to-do activity implements the loader class. All insert, delete, and update, calls are handled by the loader on its own asynchronous thread. After the database has been updated, the loader is restarted and the cursor adapter will be swapped with the new data it can iterate through and display in a list view.

To-Do Database

(Internal Storage)

Main To-do Activity

Editor Activity

**Figure 2.2** Project Flow

**4. TIMER TAB**

The timer tab of lit commit app provides a streamlined way for students to pace themselves while studying. When facing a new or difficult subject, students can feel overwhelmed when they think about how much time they need to truly understand all the material. Oftentimes students will try to cram long study sessions in which they try to study non-stop for five or six hours, but end up having a headache, feeling burned out, and end up playing video games or on their phones updating their Twitter account. Numerous studies have shown that our ability to retain information diminishes after ~25-30 minutes, so it’s best to break up our long study sessions into multiple, smaller sessions [2]. I personally have tried to study for hours on end, only to feel completely burned out after 45 minutes of non-stop studying. Since realizing my problem, I’ve been searching for a way to maximize my productivity while studying, when I discovered the Pomodoro technique.

The Pomodoro technique is a time-management method developed in the late 1980s by esteemed developer, entrepreneur, and author Francesco Cirillo while he was still a college student. This technique will help students power through distractions, hyper-focus, and get things done in short bursts, while taking frequent breaks to take a deep breath and relax. This way, students are extremely productive without feeling overwhelmed. The technique is simple: When faced with an important task or series of tasks, break the work down into short, timed intervals (called “Pomodoros”) spaced out by short breaks. The whole method consists of 5 steps.

1. Choose a task to be accomplished
2. Set the Pomodoro to 25 minutes (this is your study timer)
3. Work intensely on the task until the Pomodoro rings, then put a check on a sheet of paper
4. Take a 5-minute break
5. Every 4 Pomodoros take a longer break (~15-30 minutes)

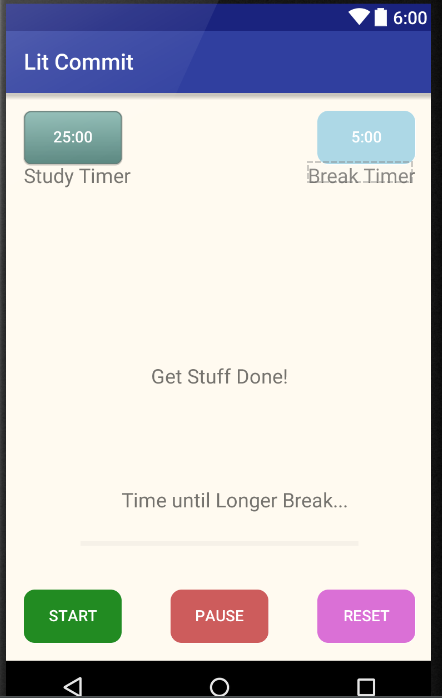


Figure 4.1 User interface of Timer Tab

**4.1 Implementation**

I implemented my version of the Pomodoro timer using two instances of Android’s built-in CountDownTimer class, which schedules a countdown until a specified time in the future. One instance was used to implement the Study Timer countdown button, and another instance was used to implement the Break Timer countdown button. The CountDownTimer class handles the messy business of creating timers on separate threads, thereby not blocking the main thread. When the app first launches and the start button is pressed, an instance of the study timer is created, the study timer button starts counting down from twenty-five minutes, and the text within that button updates every second. When that timer ends, the callback method onFinish() (of the CountDownTimer class) is automatically called. Within that method, I used the RingtoneManager class to play the default notification sound, and the Vibrator class to repeat three vibrations. This notifies the user that it’s time to set aside the books and watch five minutes of mindless television.

Above the start, pause, and reset buttons there’s a progress bar that lets the user know when he’s completed four Pomodoro time intervals & therefore deserves a longer break. At this point, the progress bar is 25% filled up. The text of the button changes to “Start Break.” When the user presses “Start break”, a new instance is created, the break timer button starts counting down from five minutes, and the text within that button updates every second. When the break ends, I again use the RingtoneManager class to play the default notification sound, but this time the Vibrator class plays two longer vibrations, thereby distinguishing the end-of-break notification from the end-of-study one. At this point, one Pomodoro interval has been completed. When four intervals are complete, the progress bar will be completely filled, and the textView above the progress bar will change from “Time until Longer Break…” to “You deserve a longer break!” Now, when the “Start Break” button is pressed, a new instance of a CountDownTimer is created with a ten-minute countdown. To distinguish between the count down timer and the break timer, I simply used a boolean variable.

**4.1 Issues / Debugging**

One issue I had was at first when the user pressed the start button multiple times, that would start multiple threads and mess up my timer. To solve this, I simply used the Button class’s .setEnabled(false) method on the start button whenever the user pressed the start button once. This blocks the button from responding to touch events. At the end of onFinish(), I call .setEnabled(true) on the start button so the user is able to press start again. Likewise, I call .setEnabled(false) on the pause button and the reset button whenever they’re pressed, and call .setEnabled(true) on the onFinished() method of both buttons.

Another issue was getting the pause button to work properly. At first when I pressed pause then hit start, the timer would restart back at 25 minutes or 5 minutes because I was creating a new instance of the CountDownTimer class. In order to correct this, I added a boolean variable signifying whether the state of either timer was paused. If it’s true, I start a new instance of CountDownTimer with the old value of the old timer saved in an instance variable.

# ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

# REFERENCES

1. Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= <http://doi.acm.org/10.1145/161468.16147>.
2. Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.
3. Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= <http://doi.acm.org/10.1145/332040.332491>.
4. Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.
5. Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.
6. Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.
7. Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= <http://doi.acm.org/10.1145/964696.964697>.
8. Yu, Y. T. and Lau, M. F. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *J. Syst. Softw.* 79, 5 (May. 2006), 577-590. DOI= <http://dx.doi.org/10.1016/j.jss.2005.05.030>.
9. Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= <http://doi.acm.org/10.1145/90417.90738>.

Columns on Last Page Should Be Made As Close As Possible to Equal Length

1. If necessary, you may place some address information in a footnote, or in a named section at the end of your paper. [↑](#footnote-ref-1)