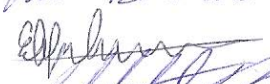
# Terp Tasker Project Proposal

## Team Red

**Submitted to Dr. James Purtilo on Sunday, November 3, 2013**

By our signatures below we declare the following are true:
1. Only the team members listed below should receive credit for the work being handed in on this assignment. We recognize that adding a name to this list after the assignment has been submitted should cause this material to be treated as a late project submission.
2. Except for resources or artifacts as identified in an attachment to this memo, and as approved by the instructor in advance, all material we submit as part of this assignment in any format is our own work.
3. Except as explicitly enumerated in the attachment to this memo, no materials, resources, software, documents or other artifacts either constituting a part of this project or used to create, prepare or transmit this project in any way bind or restrict the instructor in accessing, grading, evaluating or using this material, either in whole or in part, for any purpose.
4. Each team member grants the instructor a permanent, non-exclusive license to grade, inspect, access and use the materials of this submission.

| | Name | Signature | Date |
|---|---|---|---|
| 1. | Eric Lee |  | 10/22/2013 |
| 2. | Andrew Gao |  | 10/22/13 |
| 3. | Joshua Berenhaus |  | 10/22/13 |
| 4. | Elfalem Alemu |  | 10/22/13 |
| 5. | Matt Kicheng Song |  | 10/22/13 |
| 6. | Eyobe Biseat |  | 10/22/13 |
| 7. | Greg Clark |  | 10/22/13 |
| 8. | Emily Jones |  | 10/22/13 |
| 9. | Mark Cwalinski |  | 10/22/13 |
| 10. | Ivan Zhou |  | 10/22/13 |
| 11. | Kathryn Helgeson |  | 10/22/13 |
| 12. | Billy Wu |  | 10/22/13 |
| 13. | Paarth Chadha |  | 10/22/13 |

## Table of Contents

## I. Purpose of this Document

The purpose of this document is to describe the goals, scope, and proposed implementation of the Terp Tasker project by team Terp Tasker Red in CMSC435. This proposal is intended for Dr. James Purtilo to determine whether the product described matches his intended vision and to evaluate whether team Terp Tasker Red has met his expectations for selection of features, consideration of risks, and prediction of costs involved. Included in this document are a discussion of project goals and intended users, an overview of product features including associated risks and priorities, an evaluation of competing products, sample user case scenarios, the intended validation exercise, and an analysis of risks including security concerns and mitigation.

## II. Introduction

### *Vision*

It isn't always easy for students with a busy schedule to manage and conduct their time efficiently, and in today's fast-paced electronic world, a pen and notebook simply isn't enough. How quickly can a user determine the most important thing to do on their list? How can they access their schedule on the go? A web-based to-do manager solves these problems, and is clearly the way to go. Many already exist on the market, but they fail in one aspect: they don't help users *manage* their time. A static list of tasks is just that - a list - and beyond that, little guidance is given as to when to actually *do* something. Our vision is to create a to-do manager that actually helps users manage their time and plan tasks - more akin to a personal assistant. A user would simply input a deadline and a priority, and based on the person's personal settings, our application will help them decide when to do which tasks.

### *Project Goals*

Our goal is to create a product that functions like a personal assistant by performing various tasks intended to help the user follow a time management strategy. It should be able to assist the user in various ways. The product will be able to categorize a user's time based on the user's own preferences. For example, it should offer suggestions about what times the user should do, for instance, light intensity, administrative tasks vs. heavy intensity, coding tasks. The product should integrate into the user's existing workflow with minimal adjustments after the initial setup phase. The user should be able to define the time required and priority of each individual task. The product should then use this information to assist the user in scheduling their time in order to be more productive. The user should also have control for laying out dependencies among pending tasks so that the product doesn't recommend a task which relies on an unfinished one. The hope for this product is to assist in productivity and functionality and perform like a personal assistant for the user. Additionally, it should run on the principles of Getting Things Done.

"Getting Things Done" (GTD) is a method for managing time which allows a user to quickly capture and organize their thoughts. It also helps a user determine what to work on by organizing their to-do list by priority and time required. In general, things that can be done quickly should be done sooner, and large projects should be broken up into smaller tasks that can be completed more rapidly. Terp Tasker will be able to cater to users of the GTD methodology in two ways. Firstly, it will suggest tasks to work on based upon priority

and time required. Secondly, it will allow users to place tasks in different categories in order to stay organized.

**Project Model**

Terp Tasker is a collection of tools that will help the user categorize their time, organize their tasks by types of time and project, quickly "context switch" between two projects, view all information related to a single project, and determine how they should complete tasks based on time required, level of difficulty, and dependent tasks. First, the user will be able to categorize their time into "context blocks" based on the level of productivity they expect to work at during a certain time of day on a given day of the week. Second, the user will be able to assign tasks to difficulties, or contexts, and to projects, or categories. Thus, a user could tell Terp Tasker that they are most productive at 10-11am on Thursdays, and the application will open a context view that shows all high difficulty tasks at 10am on Thursdays. Third, the user will be able to tag tasks, events, contacts, call history, emails, and text messages with their associated categories, so all communication and upcoming work for a single project can be viewed on a dashboard. Finally, by allowing users to enter more information about each task than just a name and deadline, the application can determine how much time needs to be scheduled, including buffer time. Terp Tasker will also keep the user aware of upcoming deadlines and events with reminders. Beyond simple notifications, it will also keep the user aware of how they spend their time with follow-up emails about how they spent their time, the deadlines they met or failed to meet, and suggestions for how better to organize their time. Finally, Terp Tasker will easily integrate into any user's workflow by being available as a website, run from the virtual machine Vulgarity and available at TerpTasker.com, and as an Android app. Basic data security will be ensured on the server. For users who can install their own server and wish to ensure further security, the application will also be available as open source code with a configuration file to allow the user to run their own service. Below, we further describe the terms we will use throughout this proposal, the target audience for this application, and how we have organized this document.

***Glossary of Terms***

**Tags**: Users can assign two types of tags to the data that Terp Tasker aggregates: one tag - ‘category’ - can be applied to tasks or events, while the second tag - ‘context’ - can be applied to tasks and blocks of time called context blocks.

**Categories**: Tags which define a project, class, or type of activity - for example, “Research”, “Linear Algebra”, or “CMNS Grant Committee.” Categories are usually called projects in other productivity applications.

**Contexts**: Another type of tag for tasks which describe their effort level - for example, a particularly strenuous task can be attached to a “3 Cup of Coffee” context.

**Tasks**: A to do item with a start date and due date which can be assigned a category and a context. Tasks can be recurring.

**Events**: Portions of time which denote an activity, can have an associated category, and can be recurring - for example, a user could schedule the event “meet with Dr. Purtilo” on the calendar from 9AM-10AM on October 22nd and assign it to the category “CMSC435.” Events, unlike tasks, do not have a due date or require Terp Tasker to recommend it for completion.

**Context blocks**: Portions of time which are assigned every week a certain context - for example, a user can state Wednesday's 2-4PM as a "3 Cup of Coffee" context - meaning Terp Tasker will recommend relevantly tagged tasks for completion during that period.
**Personal assistant**: A tool which not only allows a user to list tasks and events, but reorganizes and recommends them based on their priority and deadline.
**OwnCloud**: An open source platform on which users can host their own cloud service to store files, contacts, calendar events, and other types of data.

### *Target Users*

We will focus our product's features towards college students. We will specifically be targeting University of Maryland undergraduate students for the purposes of validation testing. Students are the largest user base that would benefit from a tool that allows them to manage time effectively and succeed both in their current courses and in their future jobs. College students are young and adaptable to new technology, but generally have poor time management skills, so an easy to use personal assistant fits their needs perfectly. Many students already use multiple existing products to manage their lives; our product combines the best features from these products into a single time-management tool. It is also important that students learn good time management skills that they can then apply to their careers; our product would help them adopt these strategies before they enter the workforce.

### *Document Organization*

We will begin this proposal with an overview of product features. We will then explain individual features in greater detail with sketches when relevant. We will also consider alternatives to certain features as discussed with the client and address why we have chosen one option over another. Next, we will evaluate seven other to do list applications, analyzing their strengths and weaknesses and discussing what differentiates our product from these competitors. We will provide a sample image of each competitor's interface. Then, we will present three user case scenarios to demonstrate how our feature set aids those in our user base, as illustrated by sketches of the complete interface. Next, we will list potential risks, costs, and design challenges we expect to encounter while implementing this project, with special consideration given to security concerns and mitigation. Finally, we will enumerate the deliverables of the project, outline the intended schedule for project completion and present the validation exercise that will be used to measure the success of this project.

## III. Features

### *Overview*

We selected the features described below to best implement the "Getting Things Done" (GTD) productivity strategy. First, tasks have an associated context to denote difficulty, and tasks can be given an estimated time to completion so that the user's to do list can be organized by both types of work time and time required. Second, the task, event, and context block views and edit interfaces are simple and easy to access from the sidebar so that the user can process tasks quickly. Third, all tasks and events can be tagged with categories to help users quickly catch up on tasks for a particular class/project through a

category view. Finally, tasks can be broken up into subtasks with hierarchies that indicate that one task depends on the completion of another.

In the following pages, the features of Terp Tasker are described as part of four components: platform, inputs, views, and processing. We then discuss the tradeoffs we made when deciding on which features to include to best meet our project model. Costs for each feature and risks for the product overall are discussed later in the "Risks" section.

### *Platforms*

***Targeted Platforms:*** Users will be able to access Terp Tasker through a website and an Android app. The website allows the user to manage all of their data. We will host an instance of the website on our own virtual machine and will provide the code and the configuration file for our server with an open-source license so that anyone can run their own instance of Terp Tasker. The user can create a new task, follow up on tasks (i.e. mark them as being completed, failed, or postponed), manage categories and context blocks, import data from external services, and use the category and context views. As many of these features as possible will be implemented or augmented through pre-existing open source libraries; in particular, we will be using an open source calendar parser, calendar viewer, to do list, and contacts parser. Due to the smaller screen size and limited performance of mobile devices, only critical and read-only features will be implemented for the Android app.

***Interface Structure:*** Our application design is intended to provide an elegant and accessible interface. Users will be required to log in to allow us to access their server-side data. We will provide account management tools which allow the user to reset their password, modify their settings, and delete their account. All data will be stored in an encrypted format on a database on a server hosted on our virtual machine. We will also provide an administrative control panel to deactivate or delete accounts. The website will consist of a main panel and a sidebar that gives the user access to creating a new task; viewing the calendar so that they can add, modify, and delete events and context blocks; all category views; all context views; and a view for each type of data where users can view and tag emails, contacts, call history, and text messages. The Android app will only allow users to view the calendar, context, and category views. Both website and app users will receive notifications suggesting important tasks to complete now.

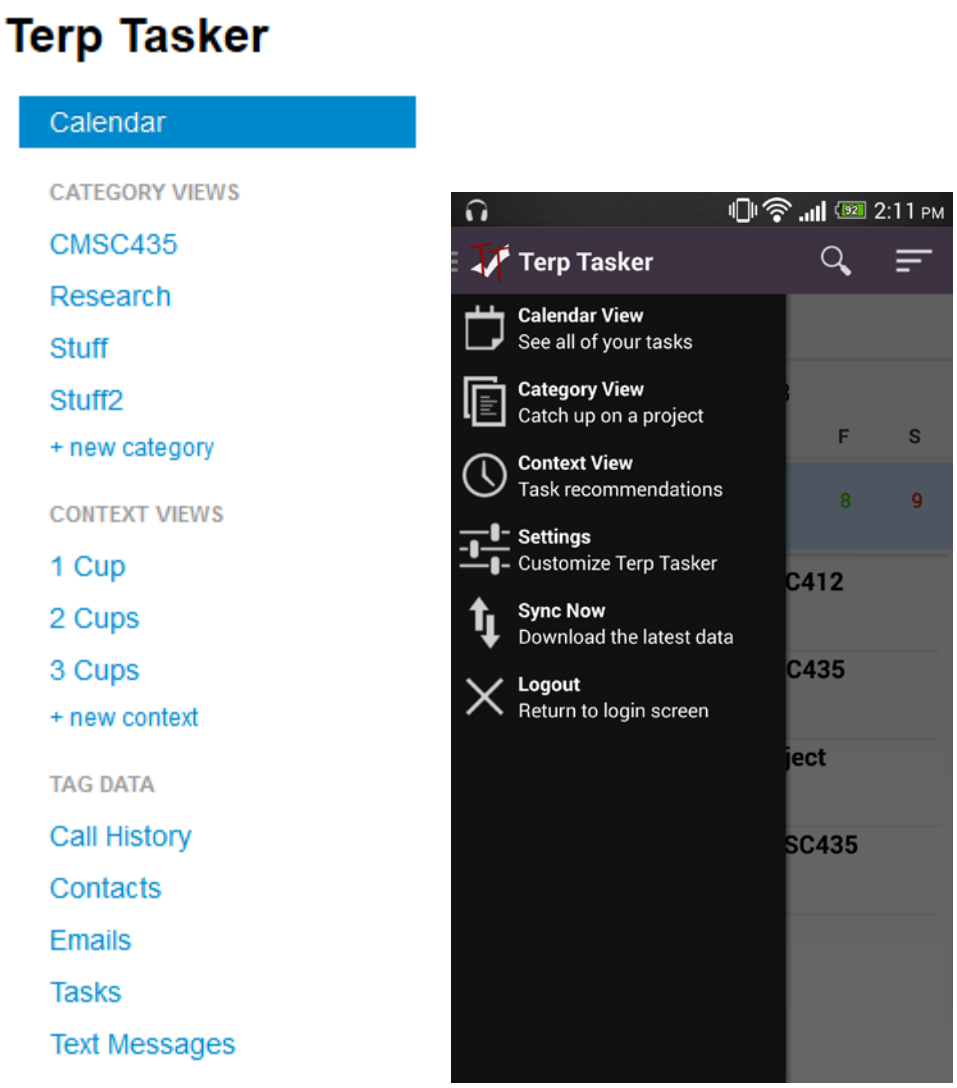



*Figure II.i: Desktop website sidebar menu (left) and Android sidebar menu (right).*

***Input***

***Tasks:*** From our website, users can create, edit, and remove tasks. A user can specify the name, start and due date, category, context, and priority of a task, as well as the amount of time the task is estimated to take (in 15 min intervals, for simplicity). By default, there are three contexts to choose from - 'one cup of coffee' indicating a low-effort task, up to 'three cups of coffee' indicating a high-effort task. Users can edit these and define their own contexts. Task deadlines can be recurring. A task can be broken up into several dependent tasks. Prior to the due date of task, Terp Tasker can send reminders in various forms (emails, text messages, app notifications). As the user works on a task, they can report on its progress by decrementing the amount of time remaining for that task until it reaches zero (completion). After the due date of a task has passed, the user can indicate whether the task has been completed successfully, postponed, or is incomplete and no more progress will be made (i.e. failure). On the website, tasks can be added from the sidebar or from the task list, context views, or category views.

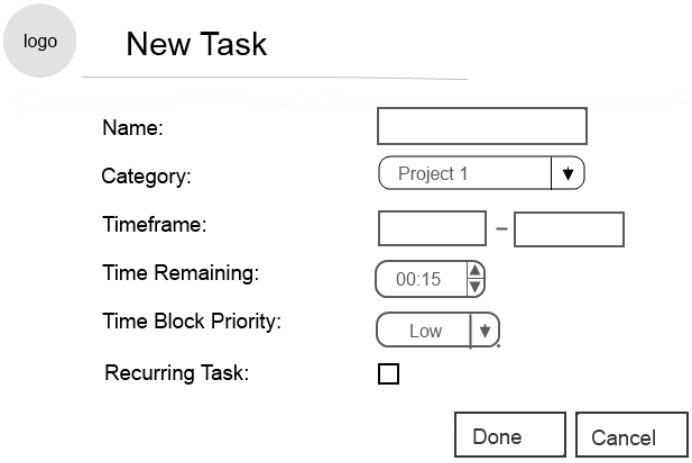



*Figure II.ii: Task creation menu.*

***Events and Context Blocks:*** The user can also add events and specify their name, start and end time, location, recurrence, and category. The user can also add context blocks and specify its context, start and end time, and category. Context blocks repeat on a weekly basis, but have a start and end date of recurrence so that a student user can reschedule all their context blocks at the end of a semester.

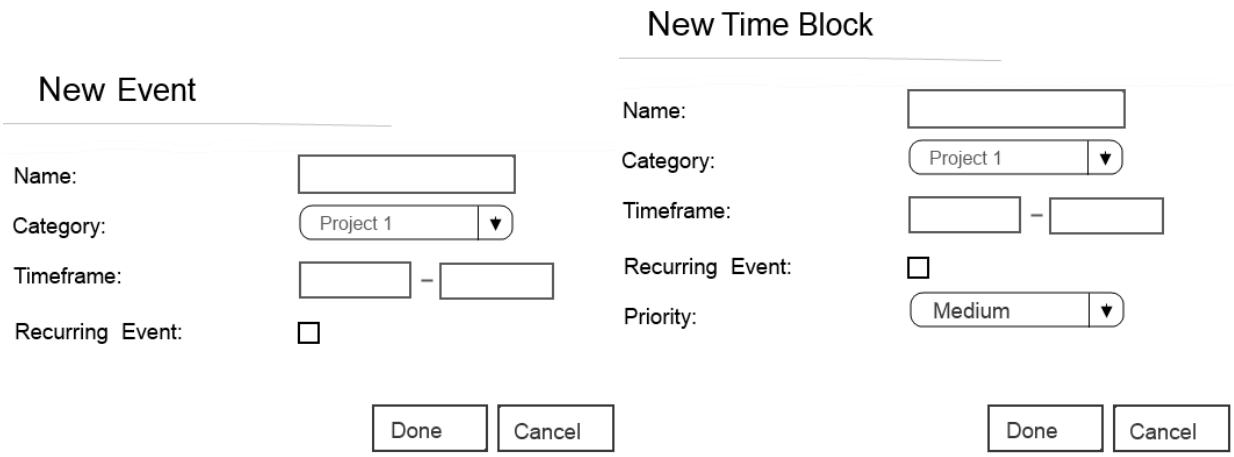



*Figure II.iii: Event and context block creation menus.*

***Data sources:*** The user can import various types of data from external sources and associate them with a specific category. Terp Tasker will accept the iCalendar file format for importing events and the vCard file format for importing contacts. Therefore, the user will be able to transfer data from any external service, including OwnCloud or Apple, that exports to the aforementioned file formats. Call history and text messages can be imported from devices running the Android platform. Emails can also be forwarded from external email clients to the Terp Tasker mail server in order to be tagged to a specific category upon a future login. All imported data can be tagged with a category and will appear in the view for that category.

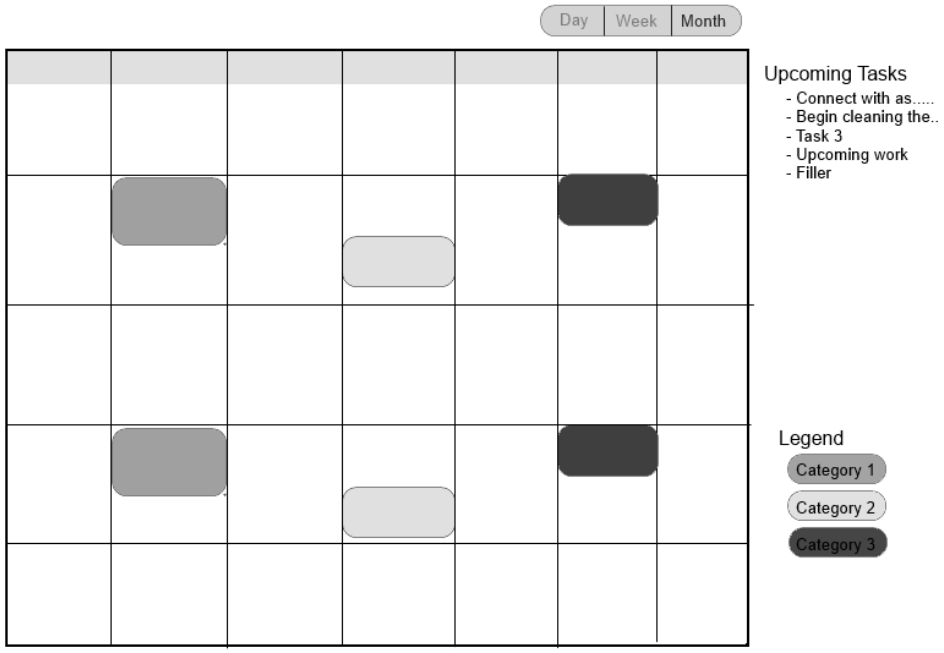
***Views***

Terp Tasker features three different ways of viewing data: a calendar view, a category view and a context view. A calendar view will have a day/week/month scale and an overall view of tasks and events for each time period. The user will have the ability to modify context blocks, tasks, and events across all categories in this view.

The category view will display all data that have been tagged with that category. When a specific category is selected from the sidebar, Terp Tasker presents all tasks, events, contacts, emails, call history, and texts that are associated with the selected category. This will most likely be used when the user wants to catch up on a specific category by viewing all information relevant to that particular type of category. During scheduled events, Terp Tasker will default to view of the category associated with the event when opened.

Finally, the context view is visually similar to the calendar in that dates are displayed, apart for one large difference - tasks are shown based on how the user has setup his/her context blocks. For example, a user can specify 12-3PM on Wednesdays as a "3 Cup of Coffee" or 'high-productivity' context. During scheduled context blocks, Terp Tasker will default to the relevant context view when opened

In both category and context views, tasks are presented in rows. The user can show or hide dependent tasks in a manner similar to computer directory navigation. The background color of a task will be a color from a gradient that indicates that task's priority and deadline. A task will also have associated icons to mark its success/failure and for deleting it from Terp Tasker.



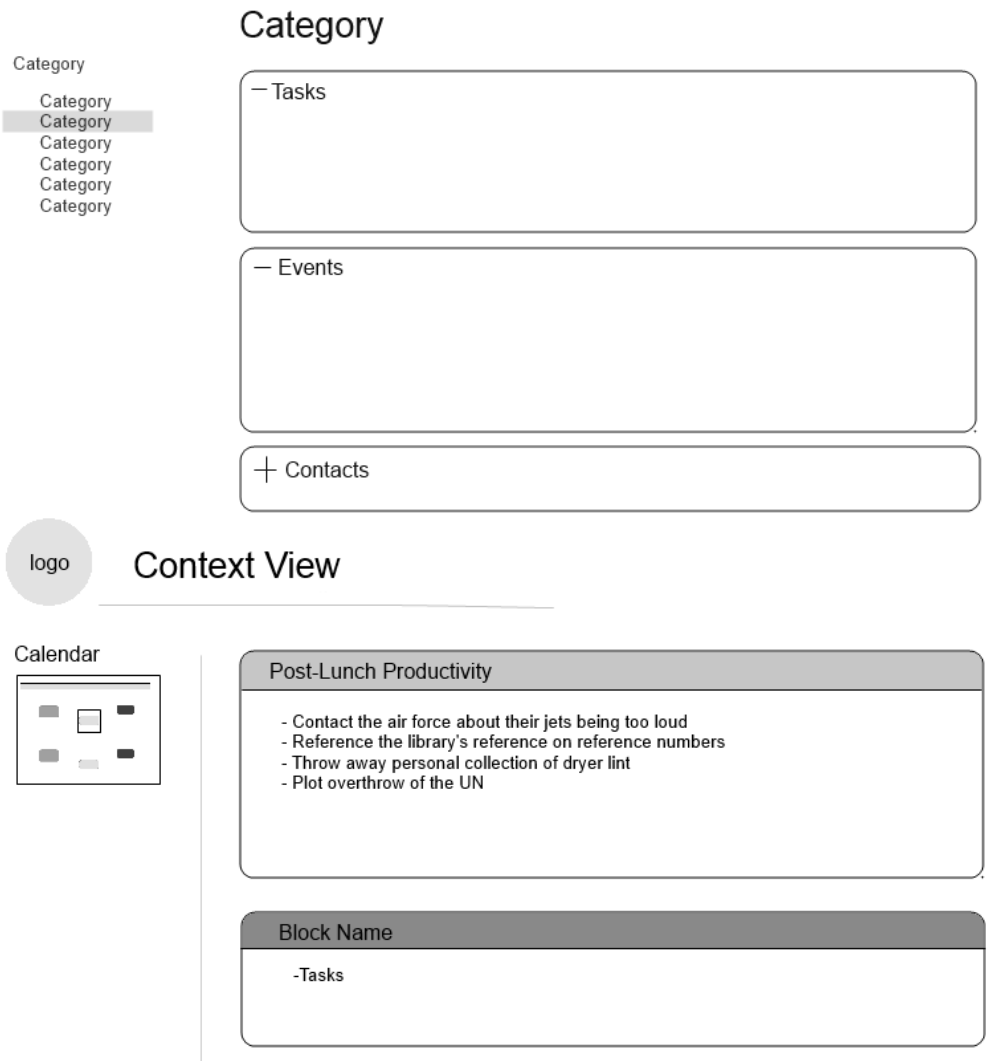



*Figure IV.iv: Calendar, category, and context view screens.*

***Processing***

***Reminders:*** In the settings menu, the user can set standard rules for three types of notifications: reminders, follow ups, and check-ins. Reminders notify the user via email, app notification, or pop up message that a task is due, an event is beginning soon, or a context block has begun. From this reminder, the user can choose to open the task, open the category associated with the event, open the context associated with the context block, or delay the context block. Follow-ups ask the user via email to update their progress on certain tasks and to schedule more context blocks if necessary. Check-ins inform the user via email about their progress so far on tasks, upcoming deadlines, and upcoming events. Reminders can be enabled, disabled, or set at particular times from the edit menu of each context block, event, and task. Follow ups and check-ins apply to the entire application and can only be enabled or disabled from the settings menu.

***Habit tracking and contingency plans:*** As the user adds tasks and logs their completion, Terp Tasker will track the success rate of tasks and completion of context blocks in order to

make suggestions with the aim of increasing the user's productivity. For example, if a user has failed to complete a task with an upcoming deadline, the application will recommend that they schedule additional context blocks to complete the task on time. If a context block is delayed, the application will evaluate if there is still enough time to complete all tasks associated with that context and will ask the user to schedule more time if needed. These personal assistant functions would be executed through notifications as described above. In addition, Terp Tasker will automatically add 20% to each time estimate plus buffer time between tasks within a context block. These can both be changed by the user in settings.

### *Tradeoffs*

There were a number of tradeoffs that we considered as we planned how to implement specific parts of our project. Here, we discuss the pros and cons of different options and explain our choices for implementation. The four important tradeoffs that we considered were the platform, the method of importing calendars and contacts, user data security, and the methods used to perform personal assistant functions.

For choosing platforms, we had three choices: a desktop app, a web app, or a mobile app. For this decision, we found that the web app would be most suited as the main platform for this project. As an organizer and a personal assistant, it's important for us to be able to access and sync server data on a regular basis. Therefore, having a web app or mobile app is more suitable in this respect. Furthermore, our main source of display would come in the form of a calendar or task information pages. Having the large screens of a monitor or laptop is more preferable, even though mobile apps would have more portability. Considering these options, we decided to implement a web app as a main platform, and a mobile app for support for more portable information.

The second tradeoff we considered was the method of importing existing calendar and contacts information. There a number of calendar and contact storage services, each with a varying level of API support, and supporting all of them would be difficult and time consuming. Thus, we considered four different options: importing from OwnCloud, Google, and Microsoft Outlook, and manual imports. After researching these options, we decided against implementing imports from OwnCloud and Microsoft Outlook. We decided against OwnCloud because although it has an API for exporting contacts, it does not have a true API for exporting calendars and relies on the user executing a number of commands from a terminal. We decided against Microsoft Outlook because its API was most suited for desktop app developments, which was not the platform we wanted to work on. In choosing between Google and manual imports, we decided to focus on manual imports. Almost all calendar/contact services have a way to export information to a standard format, so supporting manual imports covers the largest number of users. However, manually importing contacts and calendars would not allow Terp Tasker to sync with other services. We consider this to be an acceptable tradeoff as we assume that Terp Tasker would be used as a replacement for other productivity applications and that importing an updated calendar or contacts file if necessary would not be an undue burden on the user.

Our third tradeoff was deciding how user data could be stored securely. One option would be to only use data from the user's OwnCloud account, which would allow them full control of how their data was used. However, we decided that requiring an OwnCloud account and implementation would be an undue burden on potential users. Instead, we will host user data on our own server, which we will make secure as described in the risks

section below. For users who wish to host our service independently, we will provide the code open source to be downloaded and set up on the user's own server.

Finally, we considered several possible mechanisms for the application to help the user manage their time effectively. First, we assumed that users would not be able to correctly estimate time to complete a task and would not schedule buffer time between tasks. One possible solution to this is to have the application learn how long the user claimed a task would take versus how long it actually took. However, we decided that this would have too many confounding variables, including how the user estimated time based on the category of the task and how productive the user was at certain times of day. Instead, we opted to assume that the user always underestimates time to completion, and so the application will internally increase time estimates by 20% and will add buffer time. The application will also suggest that more context blocks be scheduled if the estimated time plus this additional time cannot be completed within the time scheduled. Second, we assumed that users might not know when they were best able to work in certain contexts. Again, we considered a machine learning solution to this where the application would evaluate when the user completed the most tasks in a particular context and suggest context blocks, but decided that this would require weeks of use before useful suggestions could be made. Instead, we opted to have the user schedule their own context blocks, then the application will suggest that they schedule additional time if needed. Overall, we assumed that users would make mistakes when estimating time to completion and assigning context blocks, but chose to allow greater user control supplemented by some assumptions and suggestions rather than having the application learn and develop complex protocols for assigning time for each user.

## IV. Competitor Evaluation

### *Approach*

The purpose of this competitor evaluation was to determine the strengths and weaknesses of current products within the same market as our product in order to better determine how features that fit Terp Tasker's goals are implemented well and how they could be implemented better. For Terp Tasker, we considered our direct competitors to be other individual time management tools. This means that tools such as Flow, Trello, Producteev or Bitrix24 are not included in this assessment. The features of these products do not match the features and overall goal of our product, since they are based more around collaboration and productivity within a large team and not for individuals. We analyzed both established and brand new services in order to get a diverse feel of various interfaces and feature sets. Feature sets for each service were based off of our vision for Terp Tasker. For example, our product is not focused collaboration and sharing with teams, so a feature that falls under this category would not be evaluated. On the other hand, something like an automatic scheduler or timed work session do align with our vision and will be listed as a positive. By examining seven other productivity applications, we will find examples of things we should avoid or improve for our product to be better than the competition.

***Evaluated Competitors***

***Lightning:*** Lightning is an add-on that syncs Mozilla Thunderbird's emails and Mozilla Sunbird's calendar in one place with the additions of task management to make for complete personal task managing software. Lightning does a great job of integrating its email systems and calendars under one graphics interface. It also includes many tasking parameters that help users prioritize their to-do-list and tasks. There parameters include start dates/end dates, current status, priority, percent complete, task descriptions, repeating tasks, and reminders. However, Lightning is restricted in that the software is only for users of Mozilla Thunderbird as their primary email managing system and Mozilla Sunbird as their calendar. Lightning also does not have support for related files, contacts and phone calls with their tasks. On the other hand, our application aims to provide support for multiple calendar and email sources as well as associate tasks with different files, phone calls and contacts.

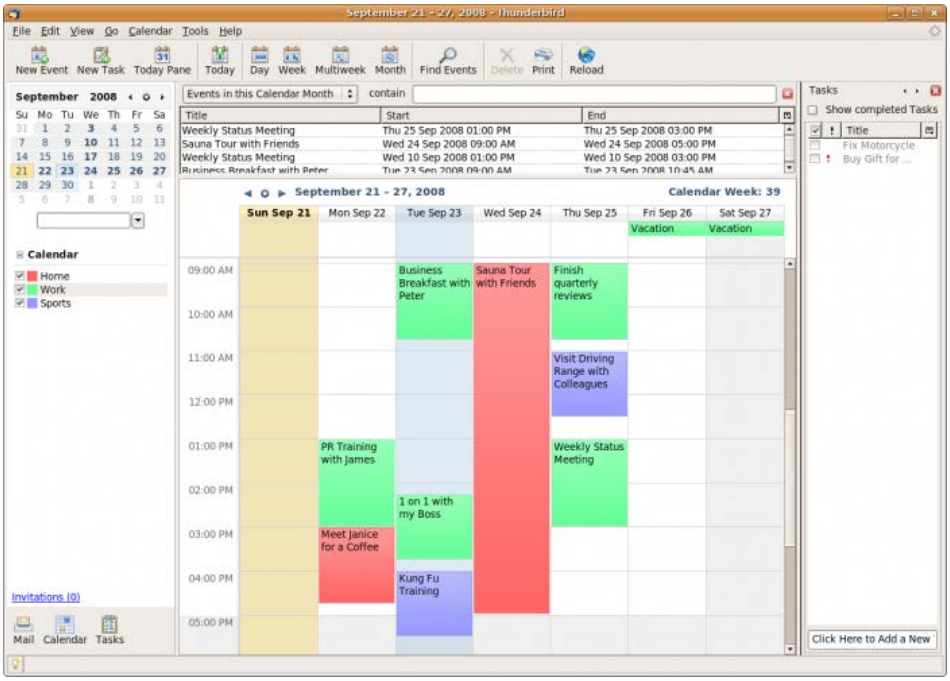



*Figure IV.i: Lightning plugin for Mozilla Thunderbird with email, calendar and tasks.*

***Apple Calendar:*** Apple's Calendar application (aka iCal) is the built in tasker app for OSX. The features on Calendar are very basic, consisting of a simple event and calendar adding and deleting system, reminders, and different views. One thing that stands out is its calendar sharing system, which allows users to share their calendar with anyone with a known email address, and even set their calendar public with a direct link. Apple Calendar is a decent tool for users who are already relatively good at planning their tasks, as it's simply a to-do list of sorts. However, it's not as great for users who are either looking for more functionality, or are poor planners. Terp Tasker on the other hand, will offer users more functionality, such as event completion/failure, and more importantly, planning based on priority.

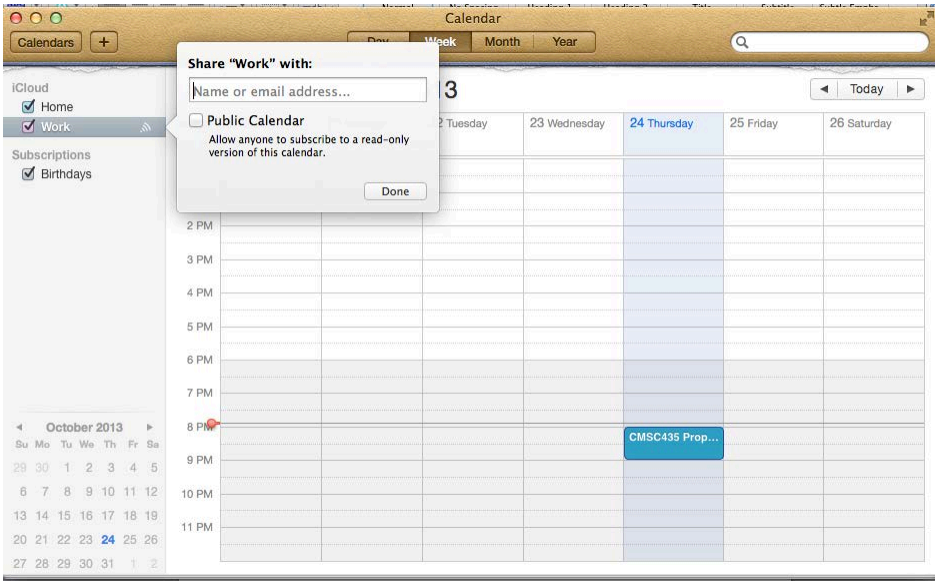



*Figure IV.ii: Apple Calendar with its calendar sharing system. Note the "Public Calendar" checkbox.*

***Google Tasks:*** This tool is an implementation of a simple task list that has access to both Gmail and Google Calendar. It has functionality that allows a user to turn emails into to-do items and add them to the appropriate date on the calendar. Users can access their lists from a mobile application, which allows them to check off completed tasks and update the lists associated with their calendar. This task management tool is good at task lists and integrating them across platform, but it is only usable by users that have a Google Account and its functionality does not do much from a personal assistant standpoint. Our product will have the functionality of these task lists but will also have more features such as specified contexts and categories, reminders, habit tracking and integration with other services, not exclusively Google.

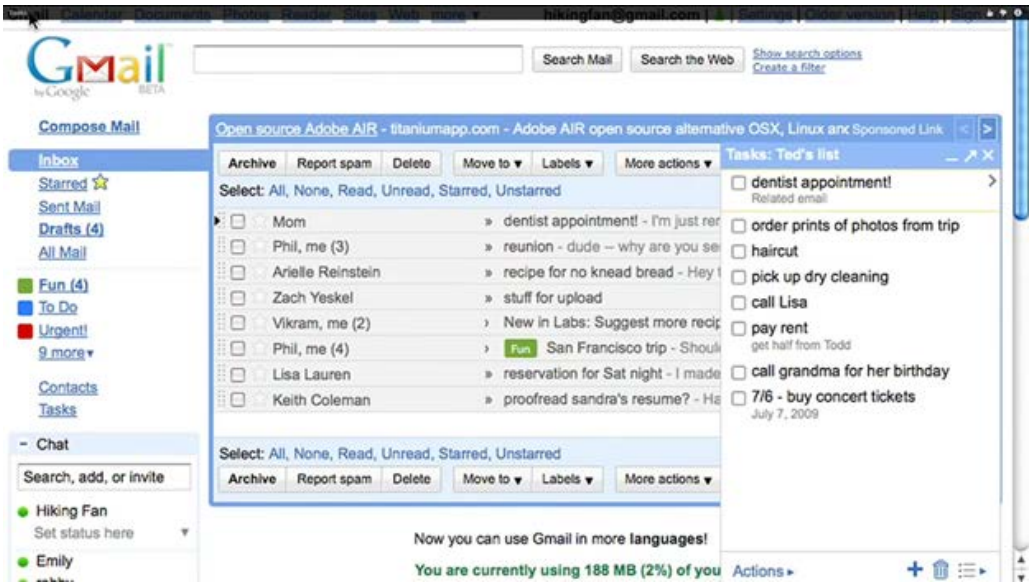



*Figure IV.iii: Google Tasks view of a task list (right side) created from Gmail interaction.*

***Nirvana**:* Nirvana is a powerful GTD based task management tool with a large feature set. Its basic features include auto-starring important tasks, accessibility from a web browser and mobile devices, contact assigning, and task creation from emails. Stronger features include contexts that isolate to-do tasks based on available time, energy, location, resources, and dates in real time; expanding any to-do task into a full project; dynamically built next-lists for step by step project completion; a logbook that keeps track of completed tasks and projects; functionality to export projects to Excel, XML and JSON. Nirvana is a good task management suite that has an expansive features list, but this fact could alienate users by offering too many features and too much in-depth functionality. Our product will limit unnecessary user involvement and maintain the core functionality that this tool offers.

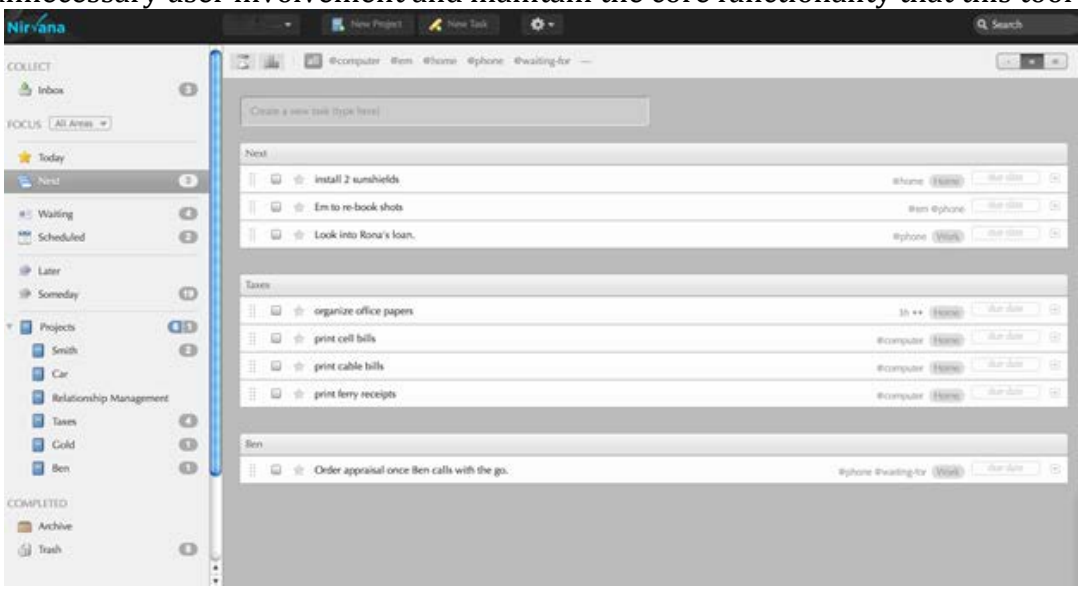



*Figure IV.iv: Nirvana layout with core features across the top, various views along the left side, and a list interface as the currently selected view.*

***Smartytask**:* Smartytask is a web-based productivity tool that implements GTD concepts. Backed by a secure server, it has functionality based around task lists. Good features include task time and task effort estimation tags for each task, an overview screen of active tasks and projects, nested goals lists, system search functionality and smart contexts based on user-given criteria. Smartytask is a good task list tool that captures the important GTD concepts through the use of flexible and task descriptive contexts. It does, however, lack any type of calendar functionality or mobile application integration, which our product will include as features.

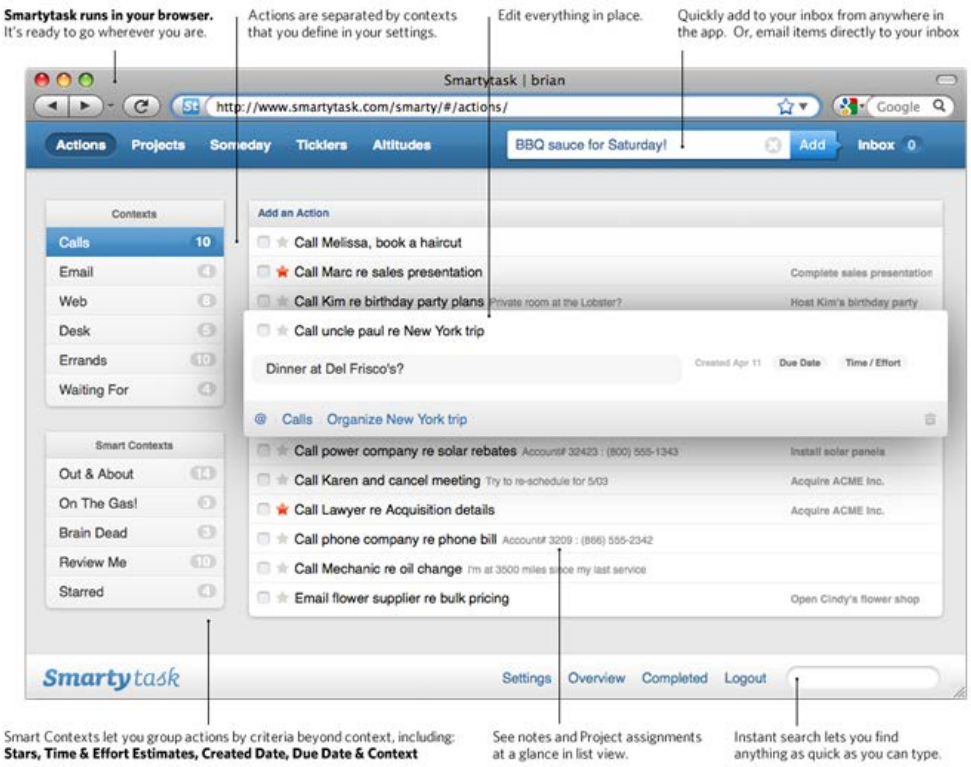



*Figure IV.v: Smartytask webpage user interface, featuring a view of a created context titled "Calls".*

***Todoist**:* This tool is a strong implementation of a to-do list. Some interesting features of Todoist are repeating tasks, HTML5 support with offline access, color-coding tasks and productivity tracking through trend charts and productivity points (karma). Mobile support and multiple platform compatibility allow a user to access their tasks and projects from anywhere, and sync with all of their other access platforms. Todoist is a good task list tool with good productivity tracking and versatile access, but it does not have any integration with a calendar, matching tasks to context blocks, or tracking of and recommendations based on task completion, which are three tools our product aims to deliver to users.

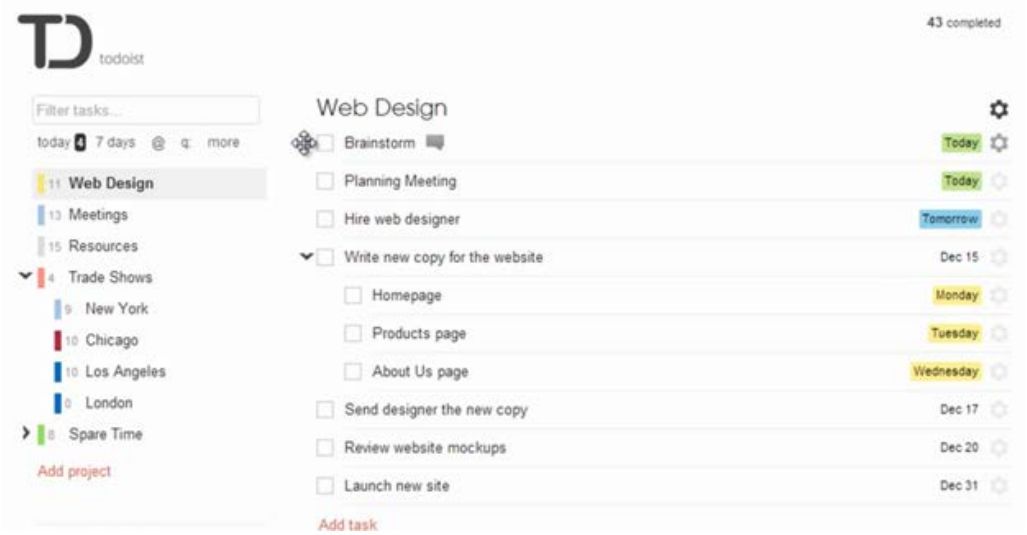



*Figure IV.vi: Todoist webpage view of a to-do list for a "Web Design" project, featuring color coding, nested lists and subtasks.*

***Toodledo**:* This tool is a feature rich task manager that, at its core, is a powerful to-do list. A few of these features include: hot lists that automatically figure out what is important right now, a scheduler to determine the best use of the user's time, the ability to attach files to tasks, subtasks that break larger tasks into manageable pieces, grouping and sorting by location, a calendar to see tasks for each day, time tracking of estimated and actual time spent on a project, and stats and graphs that track the user's progress. Toodledo is the most feature-heavy task manager that was investigated and is susceptible to the problem of having more features than a user would realistically use. Our product will revolve around core functionality that a user will find useful but not a tedious burden. Terp Tasker will also include a feature of habit tracking that is a step beyond the smart lists or statistical overview of this tool.

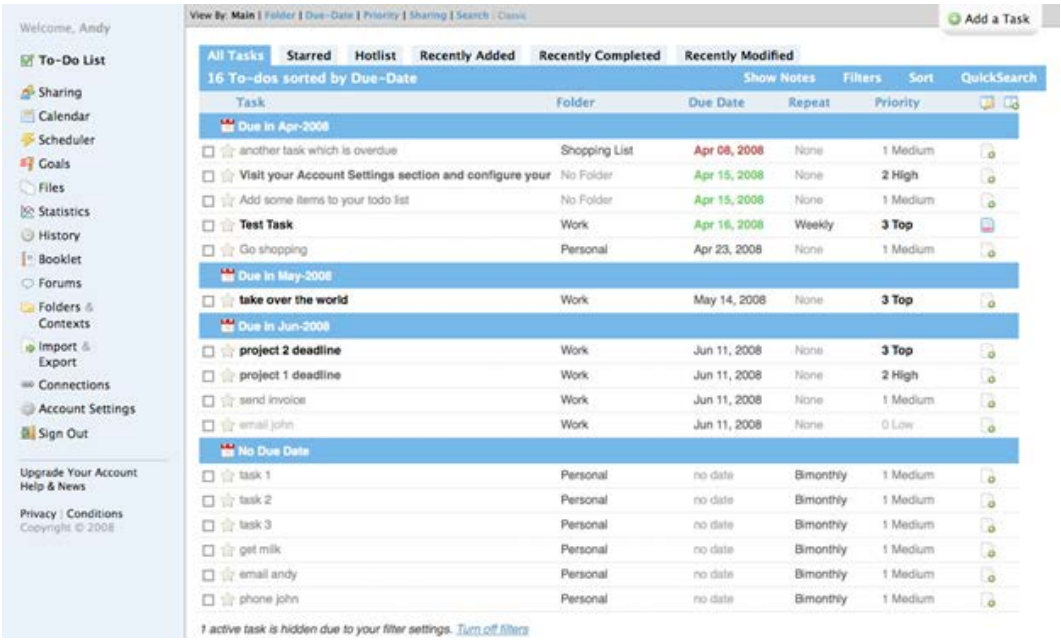



*Figure IV.vii: Toodledo list interface featuring due dates, repeating tasks, priority levels, and available views (left side).*

### *Conclusions from Competitor Analysis*

After analyzing other popular to-do lists on the internet, we were able to distill our findings down to a list of common features and problems. Almost all of the services analyzed utilized per-task contexts and tagging which was in turn used to sort and manage tasks. Task entry often had fields for deadlines, subtasks, and priority levels, and had some implementation of 'Getting Things Done' functionality. Common targeted platforms included a website for desktop/laptop access, and a mobile-optimized website for on-the-go access. Almost all services had options to 'star' tasks for importance and to remind the user via emails/texts when an important event was about to occur.

In the same vein, many similar problems surfaced when analyzing competitors. Many sites relied on solely a mobile website to support smartphones, instead of a faster, neater native app. Some competitors suffered from feature creep, resulting in cluttered interfaces, mismatched features, a steep user access curve, or a high price tag. Many services didn't interface well with calendars, or didn't support links between contacts and tasks.

This isn't to say that our competitors were all flawed - during our search we noticed some standout features. We were particularly interested in websites that implemented an automatic scheduler, repeating tasks, and duration tagging, which influenced our own selected feature set for Terp Tasker.

## V. Potential User Case Scenarios

### *User Case Scenario 1*

Suppose Bob is a busy computer science student who has a tough time managing all of his projects. In order to better manage his time, he decides to try Terp Tasker. He begins by creating an account, logging in, and uploading any of his data from outside sources that he wishes to include, such as contacts. He then opens the calendar view, adds his upcoming classes and meetings, and schedules a few weekly context blocks. Since Bob is a night owl, he assigns most of his productive context blocks later in the day and puts his less productive context blocks earlier in the day. He then creates a tag for each category. Next, he adds a few new tasks, tagging each one with its category and context. After adding a few tasks to the "three coffee cup" context, a popup message informs him that he has not scheduled enough time for that context yet, and lets him choose to either go to the calendar to schedule more time now or continue adding tasks.

Next, Bob opens the contacts page and tags his software engineering team members with the "CMSC435" category tag. He does the same for emails he has forwarded and texts and calls obtained from his phone. Now, whenever he opens the category view for "CMSC435", he will see all work and communication relevant to his class in one convenient dashboard. Then, he opens his account settings to adjust when reminders are sent. Because Bob can be forgetful at times, he sets automatic email reminders the day before a task is due. To allow him time to review his notes and travel to a new location, Bob sets text message reminders for 30 minutes before any event starts.

Using the three types of views, Bob can now get a better idea of how to spend his time and can easily access all information relevant to any class or project. He can open the calendar view to get a good sense of what he has to do this month. If he has some unscheduled free time, he can open the context view for "one cup of coffee" and finish some low priority tasks. When he walks into his CMSC435 project meeting, he can open its category view to read his recent emails to the team and add new tasks.

After putting in his preferences, Bob just follows what Terp Tasker tells him to do. Whenever one of his context blocks begins, Terp Tasker defaults to the relevant context view when opened, which gives him a list of tasks that he can work on. Afterward, Terp Tasker will send a follow up to Bob asking him if he completed the task. If he did, then Terp Tasker will ask him to update the estimated time remaining on the project. If he didn't, then Terp Tasker will reschedule the tasks or suggest that Bob carve out more time to complete his tasks.

As Bob works on his projects, Terp Tasker tracks his progress and sends Bob any suggestions if it thinks Bob can do something to improve his schedule. Terp Tasker will assume that Bob underestimates how long tasks take and so will automatically add 20% onto each time estimate. If Bob repeatedly misses deadlines, Terp Tasker will suggest that Bob has too much on his plate and should add more time to his estimates and delete some tasks if possible. Terp Tasker will also record progress during context blocks. If Bob does not seem to finish many tasks between 11pm and 12am, Terp Tasker will suggest that he move this context block elsewhere. By using all of these functionalities, Bob is able to get all of his projects done without having to pull all-nighters before his projects are due.

***User Case Scenario 2***

Now, let us look at a more specific example. Suppose Alice is a senior student applying to graduate schools. She decides to run her own instance of Terp Tasker. She downloads the code from the repository, sets up server requirements such as PHP and Ruby to match those listed on the configuration file, and follows the instructions to create a new account for herself. Next, she downloads the APK file for the Terp Tasker app and sideloads it onto her Android phone. Now, she can run the service from her own computer and can sign into her instance of Terp Tasker from her phone.

She opens her Terp Tasker account on her laptop and adds the category "Applications". She then adds two tasks to this category: "write personal statement", which has a "three cups of coffee" context, takes 4:00 to complete, starts tomorrow and is due in two weeks; and "recommendations", which has "one cup of coffee" context, takes 2:00 to complete, starts today and is due in one month, and has three ordered subtasks: "ask faculty for recommendation", "enter contact information on applications", and "send thank you notes." She then opens the contacts list and tags her recommenders with the "Applications" category tag. Next, she opens her inbox and sends all the emails she has gotten from hers schools of interest so far to the "Applications" category. Finally, Alice returns to the Terp Tasker interface and adds the application deadlines for each of her schools as an event on the calendar.

Now, it is 3pm on Tuesday, when Alice has scheduled her "three cups of coffee" context block. She receives an app notification on her phone that the context block has begun, and then she opens the Terp Tasker app, which shows the correct context view. Alice starts by working on her personal statement. She opens the category view for "Applications" and can see all her tasks, emails to schools, contacts to recommenders, upcoming application deadlines, and application materials. Thirty minutes into the context block, her research advisor calls to ask her to stop by for a meeting. From the context view, she ends the current context block, and switches to the category view for "Lab Work." From this view, she can see the current status of her research, including the emails her advisor has sent about her project, research papers she has read, and upcoming tasks. Through the category view, Alice can quickly switch gears to a new project. Terp Tasker will take care of the context time that she missed by emailing her to ask her to schedule more "three cups of coffee" context blocks before her deadlines.

***User Case Scenario 3***

A few days before an important meeting, James, who uses the Terp Tasker Android app, adds the event's details to his Terp Tasker account. If he ever needs a reminder on pertinent event information, he can open the Terp Tasker Android app and click 'Calendar View'. This view displays a simple Calendar with all of James' events. In this view he will be able to click the event in question (his meeting) and see all of his meeting event's information.

Later in the day, James notices he has some free time, but isn't sure what to work on. In this case, he should click on 'Context View'. This will show him the same information as if he was on the website. If James has defined the current time as a context block, tasks of that context will be shown and recommended here. Else, if he hasn't defined context blocks, he'll be able to choose from his defined contexts and get recommendations for each.

Even when the Terp Tasker app isn't directly open, James will still receive notifications before events/task due dates based on settings chosen on the main website. Similar to, for example, Google Calendar on Android, James can opt to get a reminder before his meeting to make sure he makes it on time.

Terp Tasker is not just a calendar and to do list, but it a personal assistant. On top of being a resource James checks on a regular basis and alerting him to urgent tasks that need to be done or events that are happening, Terp Tasker provides the ability to help James create a work schedule and follow it.

## VI. Risks

Terp Tasker's main risks can be divided into three main areas: time and personnel costs, security, and user satisfaction. As we begin the project, we will be looking to take actions in order to mitigate the risks we identify.

### *Time and Personnel Costs*

Our biggest risk is time and personnel management. We have just eight weeks to design, propose, and produce a product that is both unique and functional. Each member of our team has different programming styles and levels of experience, and these differences may lead to lost time later in the development cycle. We plan to mitigate this by holding regular meetings between all team members and more frequent meetings between team leaders. Because developing one product with thirteen programmers can be challenging to coordinate, we will be splitting up the core feature set and the additional features between each team of four. This will both reduce potential opportunities for miscommunication and allow us to match team members with particular skill sets to the project components that suit them best.

During development, we will be working with many APIs and languages that are new to the team. Some of these APIs may not be well documented, easy to use, or have much functionality at all. Thus, there will be lots of variations in the time it takes to complete a specific task with these variable parameters. We have outlined a schedule that should allow for delays and have prioritized certain features so that we are certain the core of our product design is built first. In order to save time, we will also be using as many open source versions of project components as possible as long as the time saved by not having to write new code outweighs the cost of learning how to modify the prepackaged code.

### *Feature Costs*

For each of the features we intend to create, there are risks that we need to be aware of and make efforts to mitigate. Mitigating these risks will result in a more functional product with less possibility of errors during use.

Our main platform will be the website, which provides full access to Terp Tasker. There are four main risks for this feature. The first is data security, which is discussed in greater depth in the security section below. A second risk will be web standards; we must ensure that our website is up to the standardized best practices of modern websites in order to be successful. A third risk is the speed of communication between our website and resources it uses, for example, the website and the database. To ensure that backend processes do not slow down the use of Terp Tasker, we will employ efficient database queries and algorithms. This leads to the fourth risk, which is data storage. We are limited

to the available storage on our virtual machine and so will have to reduce data replication as much as possible in our database.

The Android application provides a more streamlined version of Terp Tasker for easier access on the go. However, it also has risks. Like the website, one main risk of the Android application is data security. We need to ensure that our users are not at risk when they use this product by protecting their data and information. Another issue is Android version compatibility. Our app will compile for versions 2.3 (Gingerbread) and later which covers almost all versions currently in use. A third risk of the application is the average size of the screens most likely used to access the application. They will be much smaller than the average laptop screen, so we would need to ensure we make the best of the available space on the smaller device screen.

One of the main features of our product is task entry, where the user defines a task to be completed. One risk we face is finding the best number of entry fields to provide enough information without overwhelming the user with fields they do not need. Too few fields might not make our product useful or unique compared to other tasking tools, while too many fields might discourage the user from adding tasks.

Another feature that has risks is the email forwarding. This is the feature that forwards related emails for tagging to consolidate everything related to a task. One main risk of this feature is that the server must parse the header correctly so that the email is correctly tagged and stored. The server also must be resilient to spam and malware to provide user safety and unencumbered use. A third risk for email forwarding is getting the user to modify their current workflow to utilize the email forwarding, since this feature would require the user to use their email in a different way than they currently do.

In order to ensure that the users can combine their current productivity tools into Terp Tasker, another feature Terp Tasker would offer is importing of data, such as calendar or contact information from other tools. The main risk of this is ensuring that we can import from a variety of different products, even as external APIs of those products can vary. We have to be able to handle the variations to make the different types work in a consistent manner. We would allow uploading data in a standard file format which greatly reduces (but not necessarily eliminate) the variations in data structure across various products.

We will provide three different views to see the information in Terp Tasker: calendar view, category view, and context view. The main view that will have risks is the category view, which provides the most information by showing every item tagged with this category. These items can consist of tasks, emails, contacts, events, call history, and text messages. The main difficulty with this view is showing enough information for the user to get an accurate depiction of the category without overwhelming the user with too much irrelevant information. To mitigate this, Terp Tasker can display only recent items for each data type.

A feature we would provide to keep the user on task would be reminders. The user would be alerted in some way, for example, email, texts, popups, to an upcoming deadline or event or other item that requires the user's attention. The main risk of this feature is reminding the user in a way that provides adequate information for what must be completed for this event or deadline, without appearing so frequently that it annoys the user. The user needs to be alerted in a timely, informative manner without being excessive.

To mitigate this, we will allow users to change frequency of reminders from the settings panel and will keep the length of reminder notifications to one to two sentences.

***Security Concerns and Mitigation***

Since our product is consumer-based and users will be providing us with private information about their day-to-day life, we must provide assurance to our customers that the information we obtain will remain private and secure. Public trust plays a huge role in any web service - and any security break-ins could drive potential users away from our product. We plan to combat this risk by having the team members with the most computer security experience focus on data security and perform routine code audits as the project progresses.

**Insecure Imported Source Code:** Based on the relevance/availability of existing open-source libraries, we will be importing existing code into our project to perform certain functions. In order to ensure a secure application, we will have to audit this code for the security vulnerabilities listed below and ensure it does not undermine the security of our product. We will also do an extensive search for existing vulnerabilities in this code and fix them if possible.

**SQL Injection:** One of the most common attacks against web applications is SQL Injection. This is a technique where the hacker uses the formatting of a SQL command to perform unauthorized actions on the SQL database. While the easiest way to mitigate this attack is to escape all special characters from the input, it doesn't prevent all possible permutations. Our research has found that the most common ways to sanitize input are MySQLi or PDO, which provide an additional processing layer between user input and a database.

**XSS (Cross Site Scripting Attacks):** Cross Site Scripting attacks are similar to SQL injection in that they inject unintended code into the application, causing a malicious effect. A potential example of an XSS attack in Terp Tasker is where an attacker modifies an "add task" button to redirect users to a malware infected website. The attacker could even use this attack to put malware on our own website. However, the attack is well researched, and to mitigate it, we will use guidelines written by the Open Web Application Security Project while coding.

**CSRF (Cross-site Request Forgery):** Cross-site Request Forgery attacks are one of the more targeted attacks that our website could experience. These types of attacks abuse the "remember me" feature users select when logging in and session information to create a false authentication. This will, in turn, allow an attacker to execute commands as if they were the correct user. With this kind of attack, a malicious person could add tasks, delete tasks, or change tasks or another user's account by just having the victim click a malicious link. Like XSS, the Open Web Application Security Project has a rule sheet detailing how to prevent these sorts of attacks.

**Accounts (Log-in, Account Creation, Password Storage, Data Storage):** Users are going to be creating an account and importing a lot of their own data, so we need to make sure to keep that data safe on our own servers. We will encrypt all of the passwords using bcrypt

because it is a slow enough hashing algorithm that it cannot be brute forced easily and passwords are salted, preventing rainbow tables. It also scales with the hardware capability and is easily scalable to work with meager hardware for a slight cost of security. Since bcrypt is built in to PHP5 (> 5.5.0), it will be fairly easy to implement.

**Man-in-the-middle Attacks (SSL Certificate and Encryption):** A Man-in-the-middle-attack is when an attacker creates a fake internet access point and looks at all of the packets that are sent through their network. This could allow someone to see all of the information that a person is putting into our program which could be personal. A common way to mitigate this attack would be to have an SSL Certificate to let the user know that they are talking to the server without someone looking in on the data. We can obtain a free SSL certificate from StartSSL, and modify our Apache configuration to force users through HTTPS.

***User Satisfaction***

The last risk we will discuss is user satisfaction. Again, because we are a consumer based product, we must stay customer driven. Our success ultimately depends on how well our product is viewed by the customer. To achieve this, we will test and validate our product's features and usability against potential users and improve our product based on our feedback to ensure a polished final project. Satisfaction and usability will be measured using the narrative and satisfaction survey as described below in the "validation exercise" section.

## Deliverables

***Deliverables Checklist***

- ✓ Entire code base for website and Android app available on SVN
- ✓ Configuration file describing necessary libraries and server settings for running an instance of Terp Tasker
- ✓ Help files which describe how to set up an instance of Terp Tasker, including creating accounts
- ✓ Instance of Terp Tasker website running on virtual machine Vulgarity and available from TerpTasker.com until the end of the fall semester
- ✓ Android app APK file available to safely sideload
- ✓ Traceability documentation: engineering logs from all team members and meeting notes
- ✓ Code and results for functionality and security tests
- ✓ Validation exercise: all elements used in and produced from user testing, including consent forms, narratives used, collected data and observation from the test sessions, and surveys and use logs completed by the participants
- ✓ Open source MIT licensing of entire code base and support files

***Proposed Schedule***

| **Date** | **Action** |
|---|---|
| Nov. 3, 2013 | Submit final proposal |
| Nov. 3-7, 2013 | Get proposal "green light" |
| within 4 days from "green light" | Submit cost analysis (per group of four) |
| Nov. 14, 2013 | Demo version of all features completed |
| Nov. 21, 2013 | All features implemented |
| Nov. 25, 2013 | Alpha release (walk through) |
| Nov. 25-29, 2013 | Perform final validation exercise |
| 12:29PM December 5, 2013 | Submit deliverables as agreed upon in the approved proposal |

***Validation Exercise***

**Exercise 1**

In order to measure the success of our project, we will first perform a usability test with volunteers from our target user base. First, we will solicit 5-10 volunteers from the University of Maryland undergraduate student body to test our product. We will take volunteers from a range of majors in order to get a good representation of our target user base. To gain a comprehensive feedback, we will seek both individuals who actively use other time management tools and those who do not use such tools.

During the test, we will give each user a narrative explaining what they should attempt to accomplish using Terp Tasker, a link to our instance of Terp Tasker, an APK to be sideloaded onto their own phone, and a link to the help files. The narrative will ask them to create an account, create a new category, add a recurring event, add a task and a dependent task, tag a text message, create a context block, view a category, check off a task, view the context and category views, add events and projects, and tag emails and contacts. In addition, after interacting with the website, volunteers will be asked to use the Android app to sync data from their phone and open all three views.

We will also find one volunteer that has the technical skills to install their own server and will give them access to our open source code repository and the configuration file. This narrative will ask the volunteer to install an instance of Terp Tasker on their own server and to use the administrative control panel to delete an account.

We will measure success based on four metrics: the number of times the user could not complete a narrative item or made a mistake due to our interface, the number of times our product outputted an error or failed to do what it was intended to do, the amount of time it took for the user to complete the narrative, and user satisfaction after using our product. Errors, failures, and time will be measured by a silent observer or by video

recording of the test session of the user consents. Satisfaction will be measured by a survey asking if the user found the features helpful, whether the tool was intuitive, and whether they themselves would use the tool. If the project is successful, we expect that our users on average should be able to complete the narrative in three times the amount of time it took for someone on the team to complete it, not fail to complete more than one narrative item, and rate satisfaction 7/10 or higher on all metrics. We also expect that the user should not encounter any software failures during the testing session and should encounter at most one error.

**Exercise 2**

Our second validation exercise will be to ask these same volunteers to use our tool over the course of a 24-hour period between Monday and Friday. This period is long enough to capture the daily activities of the volunteer and test the features of Terp Tasker. We will ask each volunteer to enter their own data, including all tasks, events, context blocks, and other tagged information that they wish to share. Then, we will ask them to actively use our tool on their computer and Android phone and log each instance that they used the application, recording the application's event (such as a notification), their action (such as opening the app to postpone a deadline), and their comments (such as that they found the notification useful). At the end of the exercise, we will ask them to fill out a separate survey asking if they felt our tool helped them better manage their time, thus capturing the purpose of this tool.

**Exercise 3**

Our last validation exercise will be to confirm security and performance benchmarks. For security, we will act as our own adversary and perform commonly used attacks against Terp Tasker, such as SQL injection, XSS, CSRF, and Man-in-the-Middle Attacks. We will consider Terp Tasker sufficiently secure if it is able to successfully defend against all of these attacks. For performance, we expect that given our anticipated constraints of running our product on a virtual machine and having at most fifty people using the product at one time, any given user with a consistent Internet connection should have a maximum page load time of 3 seconds and a maximum request submission time of 5 seconds. To scale our product to handle more users, we would need to run Terp Tasker on a more powerful server, which we will not be doing for this exercise. We will measure performance and whether the features perform as they are intended using Web Application Testing in Ruby (Watir) tests.

**Conclusion**

Our mission for the Terp Tasker project is to develop a time management interface that helps students learn how to manage their time. Our target users are students, as we believe our features can help them better manage multiple projects and classes and learn good time management strategies for their future careers. Terp Tasker will, like most productivity applications, follow the principles of GTD by allowing users to tag relevant data, view all data relevant to a specific category all at once, and organize their time into blocks to complete all tasks by their deadline. However, Terp Tasker will take into account more features of tasks than most productivity applications, including their difficulty, estimated time to completion, and dependent tasks. Terp Tasker will also help users make

good time management decisions by including additional personal assistant features, including allowing users to assign time to different types of contexts, suggesting tasks to complete, sending reminders of upcoming deadlines and events, adding additional time to estimates, and checking in with users to see if they are following their own time management plan. We also aim to make this interface as convenient as possible, integrating into the user's existing workflow with minimal adjustments after the initial setup phase by having automatic notifications and recurring events and context blocks. We will measure success of this project based on our mitigation of risks, including potential security threats and limited time, and user satisfaction and success when using our interface. Overall, Terp Tasker will provide a new approach to individual task lists that will help students learn to better manage their time.

**Appendix: Notes from Meetings with Dr. James Purtilo**
***8 October 2013***
Purtilo's vision:
goal: allow the user to follow a time management strategy that:
- categorizes the user's time (eg high-gear creative time, low-gear administrative time)
- makes suggestions for how best to use time (eg this 5-hour task would be best split up into 3 segments to be completed at 10am every morning)
- adapts (eg late to work because stuck in traffic)
- incorporates seemlessly with their current work flow & usage pattern
- doesn't assume that the user is even aware of their own work strategies
the basic strategy should be GTD
- handle each task only once (eg don't check your email when you can't reply to it)
the algorithm should perform like a personal assistant

Purtilo mentioned that the tool should determine if a task is relevant (eg by comparing an email to your list of assigned tasks), assess the costs and overhead of a task, and allow you to lay out the steps. He also suggested that the tool should recognize if your load is too high and level the load and create a contingency plan in case something falls through.
Ways we could handle this:
- allow user to determine relevance by themselves in their project/category view
- allow the user to assess the costs/overhead of a task themselves by entering an estimated time for a task
- allow tasks to be part of a timeline by allowing dependencies on other tasks
- schedule tasks with break & buffer time
- allow users to check off if they completed or failed a time block, then move it to another time as necessary
Questions to ask:
- should the program be automating any of those above things?
- should the program be learning from the user to move time blocks around or suggest that they add more buffer time?
- could a follow-up email and/or stats accomplish this?
- do we need any of these things, or are they low priority?

platform: a family of tools for a family of tasks
a desktop tool with full features and an android app that has less functionality
desktop tool: could be an OwnCloud app, a client-side app (like iCal), or an outside interface that pulls from relevant APIs (see "tradeoffs")
android app: eg calendar, reminders, task list, different views, but perhaps won't allow tagging

features:
task entry: same as we described, except add an entry to categorize by different types of time (eg is this task mentally taxing, boring administrative stuff, etc)
we could either design our own names for types of time, let them make up their own names for time, or just give them a sliding scale of how difficult a task is

be sure to include dependencies

data: we should consider incorporating emails, call history, texts, calendar, & files
users should tag on their own during "administrative" time
emails can be incorporated via an email service or a plug-in (see "tradeoffs")

views: basically what we described, except:
instead of having a task view that lets you pick a productivity technique to use, it would instead have a task view that lets you pick the kind of time block you are currently in (eg show mentally taxing or creative tasks during the 'high-productivity' time slot of the day)
the objective of the project view is to context switch

iterations:
we need to test barriers to entry for all of our proposed features and APIs
- we will present two sample user cases & mock-ups to Purtilo at the next meeting, then will create more as needed
- we will present how we tested various APIs to Purtilo at the next meeting to discuss tradeoffs

trade-offs: we need to address in our proposal when we had two options and chose one over the other
website: this has less data access & security than writing an OwnCloud app, but gives us more control and a larger user base
email server rather than plug-in: a plug-in could integrate more seemlessly with the user's work, but would restrict us to one platform (this feature is not valuable enough to dedicate time to making multiple plug-ins) and might be more intrusive than intended (eg a popup asking the user to categorize an email every time they write one)

risk factors:
problems with different android OS versions
owncloud has syncing issues

validation exercise:
automated tests are expected in the deliverable, but are not part of the validation exercise
we will be meeting with him as a team after Dec 5th to do a full code and structure review during that week, we will also be doing user testing in the field

user testing:
we need to identify a target audience that we can test
testing criteria: can multiple users each complete a task...
- given a bare-bones narrative
- from start (eg installation) to finish (eg loading an email, adding a task, setting time blocks)
- under a certain time limit
- with few mistakes and no total failures (from them or the software)

- with high user satisfaction as determined from a post survey

***10 October 2013***

OK:

project view

phone as mostly read-only, not a planning tool

phone/desktop would give reminders for time blocks, allow them to be postponed & rescheduled later

user base of students to get them started early on good business practices

issues:

calendars: break with recurring meetings, edits, time zones

barrier to entry into "steady state" of not needing to perform maintenance on time management shouldn't be too high

personal assistant discussion:

adaptation: crowd sourcing?

recognize when bad things are happening

suggested adding policies like premeeting task of reviewing notes

his suggested ways to solve this problem:

suite of sample scenarios

cases to train algorithm

have team members use a paper notebook to test out different time management algorithms

he seems to be looking for a big "policy model" to handle all cases and all kinds of people and schedules

perhaps we could narrow this down by only including features that UMD students would use

***22 October 2013***

- Overall setup: forcing a user to have an OwnCloud account is not sufficient argument (they've to have an account for our server too). But we can argue that the overhead of the user maintaining a server is too much. If that's the case, we have to talk about how we will maintain the server, how can the service scale as a business, and how a reasonably tech-savvy person can install their own instance of the program. The validation exercises would also have to be modified accordingly to include how easy it is to download from a repo and install the system.
- We should a deliverables section outlining everything we will submit: instructions, code, manuals, validation exercises, etc…
- Pick a name for the product. Scarlet ledger?
- Have a clear model that is simple and expressive. It makes coding easier. How do tasks relate to each other?
- Purtilo sees the desktop service as the planning side and the mobile as the reminder and simple feedback. e.g. can't make it to some task because of flat tire, say so with mobile device, all the readjustments and contingency planning occur in the backend
- Competitor evaluation: be specific on what you're doing better.
    - look at the following: tasking in the Apple world, Lightning (a thunderbird plugin)

- Recommended getting the original version of the GTD book. Mention the core ideas of the book and how our product relates to/follows them.

Questions

- meeting notes in appendix? up to us, (yes) (becomes a citable document - think final exam)
- captions for figures? up to you (yes)
- remove risk assessment? NO, put in proposal
- feedback before grade? YES
- custom coversheet? YES

General notes:

- Sentences are you friend. Don't do bulleted lists. Write in a crisp and substantive manner. Tell the punchline, provide support, and come back to punchline. Order of paragraphs should reflect line of reasoning. Use transitions.
- Don't talk around the program. Say directly what it is. Task manager? OK but what's your edge?
- Don't forget the little things: date and document version number, page numbers, section numbers
- Be consistent, use the same terms throughout the document.

***24 October 2013***

All of the improvements below refer to our proposal version 10.24.

- We must not use sentence fragments and clearly utilize a top-down approach that introduces concepts/things at the right time.
- We should specify exactly what features we'll be delivering at the end of the semester. Move away from the priority model.
- We must make it more prominent that our instance of Terp Tasker is a demo and state the requirements for someone to rollout their own. The validation exercises should be modified to measure how easy it is to install Terp Tasker.
- Look into and include in the proposal evaluations of the Lightning thunderbird extension and how tasking/productivity is handled on Apple devices and programs.
- Specifically mention that our product will be submitted to the team SVN. In addition, if vetted for release to the public, specify the open source license that will be used.
- State whether Purtilo will have to sideload our Android app or whether we will provide an Android phone for him (in the former case, state the steps we've taken to ensure the safety of the app)
- Include wireframes in the proposal
- Include a model

The following are Purtilo's own notes from when he was going over the proposal:
"*jumping into glossary is confusing, give a clearer model.*

*best example of context issues in writing: page 5, start of overview. immediately starts "these features...." WHAT features???*

*prioritized list of features otherwise a good idea, but yourpromise/proposal to me is to do "as many as possible". Not going to fly.*

*not nearly enough comparison with existing products. this will be evaluated on my site - do you think it would be useful for you to tell me how my specific configuration will be done better?*

*delivery manifest: base code not made open source until we vet it - and the repo must be our SVN. (if the code is not there, then it does not exist.) android app needs to be either safely side loaded on my device or you need to provide a device. As written, not precise. need to make clear the distinction between all code of project versus what you'd provide for someone to stand up on their server. what are installation directions? if you do this as a service on YOUR host, then how are you handling account management? needs to be realistic.*

*this document proposes an inadequate validation exercise.*"

***27 October 2013***

The following comments are from an email reply and refer to proposal version 10.27.

"*Emily, just getting back in and so took a moment to skim what you sent. I see the main points where text was added from before - looks like the model, validation exercise and some wire frames. The first two have writing which seems in the right ballpark but presents as rushed, so make sure you make a pass at it to deal with (what was it Josh called it? Oh yeah...) a text wall. As to the content, the model is a little more elaborated but I didn't agonize over details, so have care. I'll chiefly address the validation text though. It is beefier than before but still seems like a low bar to get over. I wonder if this really tests the installation and management tasks of the system well enough? Without committing myself on where I might pick nits, I suggest that you think about the above for at least one more pass through the text, but then let's apply Purtilo's first law ("do the right thing, or do the wrong thing but let's do SOMEthing....") rather than let the perfect be an enemy of the possible. I want to get your team moving forward on the build now. I suspect we'll find any weak points operationally as we move forward rather than by thinking much harder about this now. - Jim*"

***31 October 2013***

Elfalem met with Dr. Purtilo to clarify his comments on the validation exercises of the proposal version 10.27.

- modify server installation test to include management [in general we should also have as one of our features, an administrative control panel to facilitate the management of accounts]
- say exactly what we're testing - avoid phrases such as "for instance" and "including"
- using surveys is fine
- we may want to extend the 1 day (on separate note, say 24-hr period) logging exercise to several days to really test all features (or say we can do it in 1 day), also specify if we want to test weekdays or weekends

- we should establish a set of properties that we use to pre-screen volunteers, do we want people who already use calendars or not, or have both and ask them different questions to get nuanced results