TDLoO: Getting a Grip on To-do Lists

Michael Edward Yaroshefsky
Advisor: Professor Alain Kornhauser

Submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Engineering
Department of Operations Research and Financial Engineering
Princeton University

June 2012
I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

________________________________________

Michael Edward Yaroshefsky

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

________________________________________

Michael Edward Yaroshefsky
Abstract

Nobody wants to know that a 20-page paper is due on the day it is actually due. A completely new way of adding, scheduling, and visualizing to-do items graphically provides a superior way of organizing to-do lists by encouraging users to break larger projects into smaller tasks and set daily goals. Visual clues such as position, shape, size, and color can be used intuitively to represent important metadata such as scheduled action date, expected time required, urgency, and category. Built-in, results-driven motivation tools increase the likelihood of the user setting realistic daily goals, meeting them, and associating a positive feeling with marking items as complete from the to-do list. Modern web application authoring techniques provide accessibility across platforms and devices. This technology has implications for corporate professionals, students, home use, and personal fitness schedules.
I would like to thank all of my friends who shared information about how they organized their lives that motivated me to consider this project; my dad for getting me hooked on computers from an early age; my ORFE study buddies, without whom I might not have passed some classes; Professor Kornhauser for giving me the freedom to explore something I was interested in; Briana for her motivation and proofreading; and my parents for being my support network.
To Pumpkin
Contents

Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii
Acknowledgements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iv
List of Tables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . x
List of Figures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xi

1 Introduction 1

2 Problem Definition 6
  2.1 Why Care About To-do Lists for Managing Time? . . . . . . . . . . . 6
  2.2 Paper Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
    2.2.1 Post-it Notes on a Desk . . . . . . . . . . . . . . . . . . . . . 8
    2.2.2 The Pen and Paper Method . . . . . . . . . . . . . . . . . . . . 8
    2.2.3 A Prioritized List . . . . . . . . . . . . . . . . . . . . . . . . . 9
  2.3 Software Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
    2.3.1 TaskWise . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
    2.3.2 Ta-da Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
    2.3.3 Remember the Milk . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.3.4 TeuxDeux . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
  2.4 The Hypothesis for a More Useful System . . . . . . . . . . . . . . . 15

3 Literature Survey 17
  3.1 Psychological Background for the User Interface . . . . . . . . . . . 18
3.1.1 Psychology of Visual Perception ........................................ 19
3.1.2 Psychology of Memory ..................................................... 21
3.1.3 Psychology of Attention ................................................... 23
3.1.4 Psychology of Motivation .................................................. 26
3.2 Technology Available ......................................................... 27
3.2.1 Platforms ................................................................. 27
3.2.2 Choosing a Platform ....................................................... 34
3.2.3 Client-side Technologies ................................................. 35
3.2.4 Server-side Technologies ................................................. 37

4 Market Segments ................................................................. 40
4.1 Various Segments and Their Needs ........................................ 40
4.2 Corporate Professionals ..................................................... 41
4.2.1 Persona: Brad ............................................................. 42
4.2.2 Software Needs ............................................................ 42
4.3 Academics ................................................................. 43
4.3.1 Persona: Jeff ............................................................... 44
4.3.2 Software Needs ............................................................ 44
4.4 Around the House ........................................................... 46
4.4.1 Persona: Molly ............................................................. 46
4.4.2 Software Needs ............................................................ 46

5 Using the Application ........................................................... 49
5.1 Application Overview ....................................................... 49
5.1.1 The Calendar .............................................................. 51
5.1.2 The Projects Bar ......................................................... 52
5.1.3 The Top Bar ............................................................... 53
5.2 Adding Tasks ............................................................... 53
5.2.1 Click to Add ........................................ 54
5.2.2 Drag to Add ....................................... 54
5.2.3 Quick Add Tool ................................. 54
5.3 Managing Tasks .............................. 56
   5.3.1 Reordering Tasks and Moving Tasks to a Different Date .......................... 56
   5.3.2 Changing Task Durations .......................... 58
   5.3.3 Renaming Tasks .................................. 59
   5.3.4 Completing Tasks ............................... 59
5.4 Managing Projects .......................... 60
   5.4.1 Showing, Adjusting, and Hiding the Project Bar .................................. 61
   5.4.2 Adding Projects ................................ 61
   5.4.3 Editing Project Names ............................ 61
   5.4.4 Changing Project Colors ............................ 64
   5.4.5 Reordering Projects ............................. 64
   5.4.6 Deleting Projects ................................ 65
   5.4.7 Managing Tasks in Projects ....................... 66
5.5 Motivation Features .......................... 67
   5.5.1 Reminder Emails .................................. 67
   5.5.2 Progress Reports ................................. 68

6 Design and Analysis .......................... 70
   6.1 Missteps with Complicated Frameworks ........................................ 70
   6.2 The Ext JS Chapter of Development ........................................ 73
   6.3 Back to jQuery ........................................ 75
   6.4 Technical Challenges .................................. 76
      6.4.1 Data Storage and Synchronization ........................................ 76
      6.4.2 Undo and Redo ..................................... 81
      6.4.3 Offline Mode ....................................... 84
7 Business Model

7.1 Business Models ........................................ 89
    7.1.1 Advertising ........................................... 90
    7.1.2 Donations ............................................. 90
    7.1.3 Trial Period .......................................... 91
    7.1.4 Freemium ............................................. 91

7.2 Payment Arrangements ................................... 94
    7.2.1 One-time Fee ......................................... 94
    7.2.2 General Subscription ................................. 95
    7.2.3 Freemium Subscription with Service Levels ........ 95

7.3 Implementing a Model in TDLoo .......................... 96
    7.3.1 Running the Numbers ................................. 97

8 Conclusion ............................................. 99
List of Tables

2.1 TaskWise at-a-glance ................................................. 10
2.2 Ta-da Lists at-a-glance .............................................. 11
2.3 Remember the Milk at-a-glance ................................. 13
2.4 TeuxDeux at-a-glance .............................................. 14
List of Figures

3.1 The eye perceives similar shapes as belonging to a group. ........................................ 20
3.2 The most distinguishable colors are those that directly stimulate one of the three color-perception channels and send neutral signals to the other two channels. ............................................................ 21
3.3 The human eye has many photoreceptors in the center, or fovea, of the eye but fewer towards the edges, leading to poor peripheral vision. Subfigure A approximates the resolution of human vision, and Subfigure B simulates how the eye’s ability to distinguish letters decreases as the letters move away from the center of the fovea. ........................................ 22
3.4 Microsoft Word for Mac uses a familiar notebook metaphor, including tabs dividing sections. ............................................................ 24
3.5 The page turning animations and shading details are smooth enough to maintain the book metaphor in iBooks on the Apple iPad. ........... 25
5.1 The application as a user may see it. The tasks appear in different colors according to the projects with which they are associated. ....... 50
5.2 The calendar section of the application is the main focus of the appli-
cation. The current day is highlighted in blue. ........................................ 51
5.3 The project bar provides the user with additional flexibility for managing complicated projects or categories of tasks. The user can pick colors for each project, which are represented by the thin colored bar below each title.

5.4 The name of the list is clickable, giving the user the ability to have multiple different lists.

5.5 The quick add bar gives users an easy way to add tasks by typing them in and hitting enter. It features a syntactic analyzer to parse dates.

5.6 Users can drag tasks from one location and drop them in a new location.

5.7 As the mouse hovers over the task, a handle appears at the bottom which can be dragged up and down to change the duration of the task.

5.8 The sizes of the task correspond to the different times the user estimates they will take. By indicating the expected duration prominently, the application can help users avoid overcommitting themselves in a given day.

5.9 The application makes significant use of mouseover properties, changing the appearance of user interface items to indicate that the mouse is hovering over it. Hovering over an item can reveal functionality that is otherwise hidden.

5.10 Users can show, hide, and adjust the projects bar.

5.11 The color palette provides an easy way for users to preview colors before selecting them.

5.12 The color swatches are selected based on the psychology of color perception to be easily distinguishable.
5.13 The headings of the projects include a handle on the top left for re-ordering the projects, an icon in the top right for deleting the project, an icon in the bottom right for changing the color of the project, and a thin sliver at the bottom showing the color of the project.

6.1 280 Slides is a web application for developing presentations that uses the Cappuccino framework. It contains many features and an interface one would expect to find only on a native application, not in a web browser.

6.2 This mockup, designed in Photoshop, represents an early concept of what that application was to look like.

6.3 This basic structure was generated by Ext JS using a programatic approach.

6.4 jQuery has rapidly gained popularity in recent months.

6.5 The browser and client are separated by the internet, and they communicate using Ajax via the jQuery library.

6.6 Action tokens are sent to the controller, stored locally, sent to the server, compared against the server’s records to determine validity, stored on the server, and then confirmed to the client.

7.1 Basecamp by 37signals offers a 45-day trial period for its project management software.

7.2 Dropbox offers multiple tiers of pricing, ranging from free to various levels of premium service.
Chapter 1

Introduction

The ability to concentrate and to use your time well is everything if you want to succeed in business—or almost anywhere else for that matter.

-Lee Iacocca

The study of operations research is the study of using data to make informed decisions that will lead to a desired outcome. Usually these decisions are of a strategic nature, involving resources such as money, personnel, or raw materials. For these problems, there is typically a well-defined objective function and other parameters such as constraints that can be used to determine a feasible and, in some cases, optimal solution.

Take, for example, the so-called newsvendor problem, an operations research staple. In this problem, named for a newsvendor who must determine the optimal number of newspapers to have in stock for a given day, there is an objective function to maximize (profit) and there are resources which must be allocated (newspapers). The goal of this prototypical problem is to achieve a maximum profit, with the understanding that there is a cost of having too few newspapers (profits that cannot be earned) and a cost of having too many (the papers are useless the next day).

Another class of decisions is personal time management. For this type of problem,
there may not be a clearly defined objective function to maximize or minimize, as this could vary by the individual to whom it applies. However, there is a generally applicable goal: using the minimum amount of time to accomplish a set of goals. In other words, the goal of time management is to be productive, leaving excess time for personal enjoyment or for accomplishing more in a given day.

Part of what makes time management a worthwhile problem to consider is the frequency with which individuals are making and executing plans regarding it. At every moment, a person is either making a plan or acting on a plan for how to allocate their time. Right now, you are reading this document, thereby executing a plan you made for managing your time, and once you have finished, you will once again make a plan for what to do next or follow through with a plan you made previously. People make these decisions many times each day, so it is a pervasive problem for students, professionals, government officials, and anyone else who has free will and tasks to accomplish.

Perhaps the most important reason why time management is worth studying is because of the positive impacts on our lives that effective time management can have. When we use time effectively to bring about the results we want, we reduce stress and frustration, we become more confident, and we feel better (Mancini 25). By taking less time to accomplish the same set of tasks than if we had not used an effective time management plan, we have more time left over to recuperate and be more productive later, and we also gain a sense of accomplishment for what we have marked off our to-do lists.

Since not all strategies for time management work for all people, there are a number of different tools available, ranging from swimming in a sea of Post-it Notes to using an advanced software system. There are popular mainstream solutions, such as the FranklinCovey day planner. These planners, however, suffer from the drawbacks of any non-software system: they are difficult to modify once inked up;
they are not as portable as an app on a mobile device or web application; they are static sheets that cannot be adapted to the users needs; and there is no possibility of receiving scheduling “help” from a computer algorithm. This is where a software solution comes in.

Software-based to-do lists provide a number of important benefits. First, modifying a software to-do list is potentially much easier than modifying a sheet of paper with ink or pencil markings all over it. This is important if the user’s priorities suddenly change or something needs to be added in after the list was made. Second, a software system that makes good use of the cloud can allow for universal accessibility across a variety of systems, including a browser-based solution that can be used from any internet-accessible device and native applications for popular mobile operating systems such as iOS or Android. This is important because a user can only use his or her list when it is accessible. Third, software systems can adapt to the user in ways a sheet of paper cannot. No to-do list designer can fully anticipate all of the ways in which a user might wish to customize the system to match his or her unique needs; a student may want a column for each class, and a corporate professional may want a column for each client. Fourth, a software to-do list can allow the computer system to do some of the work for the user, such as determining the appropriate order in which to work through a to-do list or reminding the user via email of important tasks to do.

Software applications for managing a to-do list consist of three essential components: (1) user input, (2) computation, and (3) output. For paper systems, there is only user input (writing with a pen) and output (reading it back). No computation takes place, and the user input exactly matches the output. The design of software is such that the user input is operated on by the system, and the output is usually of a different, more readable and useful format.

An essential part of any application is the interface between the user and the
software. This affects the way in which the user provides information to the system and the way in which the system communicates back to the user. Arguably, the user interface is the most important aspect of any application, since without an interface, the system could not work, and with a poor interface, it cannot work well.

For a to-do list application, the user experience (a more comprehensive term than just ‘interface’) is of the utmost importance, since the value added by the system comes from the ability to organize and view priorities rather than the system’s ability to compute something. Because the most important part of the application is when the user inputs the information and reads the information back, and not the processing that’s done on this information, the user experience is the most important part of the application.

Therefore, there are three goals for this project:

The first important objective is to make managing a to-do list as effortless as possible. The user interface must be intuitive enough so that users do not need to read a guide to use the software, and it should be powerful enough that users can rapidly make changes without much need for thought. Additionally, this concept of effortlessness goes beyond just a great interface and includes features such as multi-platform accessibility, cross-browser compatibility, and responsiveness. Even if the tool performs fabulously well at organizing to-do items, if it fails at being straightforward and accessible, it will not be better than the alternatives.

The second objective is that this project must allow a user to organize to-do items in a way that makes them fit more easily into their day. Most to-do lists only strive to give users the power to store and retrieve information, but, by permitting users to organize and plan their to-do items over the course of weeks and months, this project empowers users to develop strategies for making headway that fit into their own schedule. Nobody wants to know that a 20-page paper is due on the date the paper is due. To-do lists can add value to a person’s life by allowing the user to split
up larger projects into smaller chunks and schedule when he will work on the project, rather than just making a note of when the project is ultimately due.

The third objective is to motivate users to follow through with the schedule they designed. This can be accomplished by “helping the users remember their tasks and also adding a game layer to the system whereby users are rewarded for completing them. Further, the graphics of the program can make “striking off” a to-do item a particularly rewarding action and allow users to see all that they have accomplished, allowing them to feel positive about their accomplishments and look forward to completing other items on their list.

Most computer users have already found ways to manage their to-do lists, probably without giving much thought to them. For them, software to-do lists are usually relegated to the side of an email client, a collection of virtual sticky notes, or simply a list in a word processor. In short, to-do lists are often an afterthought – an inconvenience and necessary evil. For those that are either more adventurous or serious, a number of cross-platform web applications provide more robust to-do management systems. Among the most popular ones are Remember The Milk, Todoist, Ta-da Lists, and Toodledo. These applications, however, are all incredibly similar. They are, essentially, glorified lists. Of course, a “to-do list” implies that the data ought to be represented as a “list,” but perhaps there is a better way to approach the problem of time management. Rather than create another to-do list application, why not start back at the original problem and design a better solution from the ground-up, one based on a more thoughtful consideration of personal time management techniques, modern user interface design, and the psychology of motivation?
Chapter 2

Problem Definition

*If you don’t know where you are going, you’ll end up someplace else.*

-Yogi Berra

It is important to understand why individuals should manage their time. Identifying the main advantages of making a to-do list can suggest what a new tool should hope to accomplish. Further, considering the strengths and weaknesses of currently available tools, ranging from using Post-it Notes to a software application, motivates the design of a new application that addresses these shortcomings.

2.1 Why Care About To-do Lists for Managing Time?

In its most basic form, a to-do list is a series of goals. “Take out the trash,” for example, is a goal likely to be found on a to-do list. There is an objective: for the trash to be outside. There is a success criterion: when the trash is outside. There is an action that can be taken to bring about the success criterion: physically picking up the trash and moving it to the curb. Other goals on a to-do list may include, “Send the annual report to John,” “Buy broccoli,” or “Read chapter 4.” It is important to
set goals because “you cannot make the most of your time without knowing what
you’re trying to achieve” (Mancini, 2007, p. 120).

Strictly speaking, however, to-do lists bring very little direct benefit. The main
value of any written list is that the items will not be forgotten. For most people,
this is the most basic benefit of using a to-do list; by writing down what needs to be
done in a format that can be read later, we reduce the possibility of forgetting what
we need to do. However, there’s more to time management than marking items off
a to-do list. “Effective time management isn’t the key to a completed to-do list, it’s
the key to an accomplished life” (Andrews, 2011, p. 121).

The hypothesis of this project is that to-do lists can provide significantly more
benefit to a person’s life than just reducing the chance they forget something. During
the process of writing down what needs to be done, the user should be able to assess
his or her responsibilities frankly, considering such information as the length of time
and the priority of each task. The user can then rely on this information to make
choices about when to do a certain task. Further, when realistically noting the time
length of each task, a user will be less likely to overcommit on any given day, seeing
how all of the items that need to be done add up.

2.2 Paper Options

There are a number of ways in which people can currently make to-do lists using
paper. These methods vary in their complexity and also in their utility. However, all
of these methods suffer from the challenge of modifying them after they have been
created. This is why a software application would be better suited to an organizer
that encourages the user to make changes as they are planning.
2.2.1 Post-it Notes on a Desk

These canary-yellow sheets of paper with sticky edges have become ubiquitous around corporate offices. They are cheap and easy to use. Anything that needs to be remembered can simply be written down and posted to a flat surface. The advantage of this method is that it is simple to understand, the brightly-colored notes are hard to ignore, and they can easily be moved around (by unsticking and resticking them). There are many problems with this method, though. First, Post-it Notes can easily fall behind a desk or otherwise be lost, thereby compromising its utility as a reliable device for remembering things. Second, there is limited desk space on which to place these notes, and anyone using this system is likely to have a messy desk that can make them appear careless and disorganized. Third, there’s no easy way to know what is on such a list when not at the desk. Therefore, using Post-it Notes may work for a few reminders, but it does not seem likely to serve the needs of someone trying to juggle many priorities while on-the-go.

2.2.2 The Pen and Paper Method

Another approach is simply writing to-do items down in a notebook and carrying this around, perhaps in the form of a personal organizer. This is another classic example. Once again, this method is simple to understand and use. Just write down what needs to be done, and check it off once it is done. Aside from its simplicity, this method is popular because it is portable, and it doesn’t need much explanation to understand how to use. The drawbacks, however, are that it is very difficult to move or change items once written down. This means someone using this method to organize themselves will be slowed down by copying items around on their list. Also, there is a considerable risk associated with losing this notebook or leaving it behind somewhere. Therefore, while this is also a simple way to keep track of items on a list, it still misses the mark.
2.2.3 A Prioritized List

A more advanced list-making technique, as suggested by FranklinCovey day planners, is to prioritize tasks. This is a small innovation on the pen and paper method, whereby the items on the list are sorted according to their priority (which is assigned by the user). The advantage to this method is that there is additional information stored in this list by virtue of their order (or, in some cases, the priority number written beside each task). The list-maker can then use this data to act on these tasks based on their priority. There are downsides to this approach, however. If the person acts only on the items with the highest priority first, items which might take a very long time, there is a chance that new high-priority items will be added to the list before the person ever gets to the bottom of the list, thereby leaving certain lower-priority items untouched. Further, on a written list there is no easy, clean way to reorder or modify it without copying all of the items to a fresh sheet, making changes to these lists either messy or tedious.

2.3 Software Options

There are many software options available for managing to-do lists, each with unique strengths. However, no application seems to get everything quite right. Some applications, like TaskWise, offer too many features, making the application overwhelming. Other applications, like Ta-da Lists, severely limit what the user can record about tasks. Nonetheless, these applications may have features worth considering while building a new application.

2.3.1 TaskWise

Among all of the online to-do lists, TaskWise seems the one designed most specifically for the needs of businesspeople. The site has a very corporate look, with clearly
Table 2.1: TaskWise at-a-glance

<table>
<thead>
<tr>
<th>Name and Tagline</th>
<th>TaskWise, Work Smart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Founded</td>
<td>2007 by Silversoft Solutions Ltd.</td>
</tr>
<tr>
<td>Price</td>
<td>Free or $30.00 per year for TaskWise Pro</td>
</tr>
<tr>
<td>Address</td>
<td><a href="http://www.taskwise.com/">http://www.taskwise.com/</a></td>
</tr>
</tbody>
</table>

delineated modules and sections. This application has a number of features that are useful:

1. Hitting the 'tab' button creates sub-tasks, meaning projects can be represented easily by using hierarchy.

2. The entire list can be exported to Microsoft Excel for further analysis and management.

3. The list provides statistics to help the user measure their progress and productivity.

The main problem with this list application, however, is that is overwhelming to use and gets in the way of the user achieving his goal:

1. The darker color scheme gives the application a heavy, imposing feel.
2. There are so many buttons, knobs, and gizmos that the application appears visually overwhelming.

3. Some icons are not labeled with words, leading to guesswork in usage.

This list program could certainly meet the needs of power users, those who need all of the features of this application and are prepared to invest some time understanding the application, but it is far from a joy to use and look at. This application does not encourage users to plan when they will do their tasks, only to list them so they will not be forgotten.

2.3.2 Ta-da Lists

Table 2.2: Ta-da Lists at-a-glance

<table>
<thead>
<tr>
<th>Name and Tagline</th>
<th>Ta-da Lists, It’s easy to-do!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Founded</td>
<td>2005 by 37signals</td>
</tr>
<tr>
<td>Price</td>
<td>Free</td>
</tr>
<tr>
<td>Address</td>
<td><a href="http://www.tadalist.com/">http://www.tadalist.com/</a></td>
</tr>
</tbody>
</table>

Ta-da list is a no-frills to-do list. Unlike TaskWise, this application is streamlined and simple. There are no unlabeled icons (or, for the most part, any icons what-
soever). Everything is well-labeled and intuitive. There are a number of attractive features to this application:

1. Examples are provided in each field so users know what to type or click

2. The fields are labeled with easy-to-understand, active commands rather than prompts (i.e. “Name your list” rather than “list name”).

3. When tasks are completed, there is a simple but gratifying animation as completed tasks jump to the bottom and fade out.

This application’s greatest asset is its simplicity. However, this results in a number of shortcomings for the purpose of being a reliable personal organizer:

1. Reordering tasks on the list requires changing context into a “reorder mode,” which is clumsy and time-consuming.

2. There is no way to add additional information to any task, such as date due or priority.

3. It is not possible to tag or categorize tasks.

This application is useful for those looking to jot down a simple list. It would be ideal for a list of items, like a grocery list, but it does not lend itself very well to personal organization because it is so basic. If the porridge was too hot with TaskWise, it is too cold with Ta-da Lists.

2.3.3 Remember the Milk

By far the most balanced to-do list application reviewed here, Remember the Milk (RTM) outpaces all of the others in features as well as ease of use. For example, with RTM, the user can set priorities, due dates, recurring tasks, time estimates, tags, locations, URLs, and notes. Nevertheless, these features are integrated subtly
Table 2.3: Remember the Milk at-a-glance

<table>
<thead>
<tr>
<th>Name and Tagline</th>
<th>Remember the Milk, Never forget the milk (or anything else) again</th>
</tr>
</thead>
<tbody>
<tr>
<td>Founded</td>
<td>2005 by Remember The Milk Pty Ltd</td>
</tr>
<tr>
<td>Price</td>
<td>Free or $25 per year for Pro account</td>
</tr>
<tr>
<td>Address</td>
<td><a href="http://www.rememberthemilk.com/">http://www.rememberthemilk.com/</a></td>
</tr>
</tbody>
</table>

so as not to make the application tough to use. Users do not need to provide this additional information, but they can if they want to. Tasks can even be shared with other RTM users. This application stands out for a number of reasons:

1. There are many useful keyboard shortcuts that make this ideal for power users to execute common commands easily.

2. The common language parser is superb, allowing users to indicate times with text such as “call bob next tues 8pm,” and RTM automatically parses and understands this as a relative date in the context of today’s date.

3. The day-at-a-glance and the to-the-minute reminder emails are very useful for keeping users aware of what they committed to achieve.

There are, however, a few shortcomings to RTM that could be addressed with a new application:
1. It is not possible to select a date on which an item will be done, only the date on which it is due.

2. The single list of tasks can become unbearably long and overwhelming to look at.

This application exceeds the rest in functionality and flexibility, but the interface could be improved to avoid list overload, and certain important features could be added to give users more power over scheduling.

2.3.4 TeuxDeux

Table 2.4: TeuxDeux at-a-glance

<table>
<thead>
<tr>
<th>Name and Tagline</th>
<th>TeuxDeux, a simple, designy, free, browser-based to-do app</th>
</tr>
</thead>
<tbody>
<tr>
<td>Founded</td>
<td>2009 by Swiss Miss and FictiveKin</td>
</tr>
<tr>
<td>Price</td>
<td>Free or $2.99 for the iPhone app</td>
</tr>
<tr>
<td>Address</td>
<td><a href="http://www.teuxdeux.com/">http://www.teuxdeux.com/</a></td>
</tr>
</tbody>
</table>

TeuxDeux is the only list application available that also uses a calendar metaphor for organizing tasks. This application can be used to demonstrate how valuable it is to arrange tasks not just on a list, but on many, where each day is a separate list.
Users can add tasks to these daily lists like they would add appointments to days on a calendar. The application has a number of strengths beyond this calendar metaphor:

1. The interface is remarkably clean and minimalist, so the application gets out of the way, letting the user focus on organizing their tasks.

2. The application expands to fill the width of the screen on which it is displayed, making it enjoyable to use on a variety of screen sizes.

3. It is easy to get back to today’s date and know what day is today on the calendar.

4. There is also an iOS app the synchronizes with the online system and maintains the same interface principles.

Despite these strengths, the list application does have some shortfalls, mainly a result of the developer’s focus on simplicity:

1. There is no way to specify additional information about tasks, such as due-dates, categories, or priorities.

2. Only five days are shown at a time on the calendar.

3. It’s not possible to edit the items in-line, which is not intuitive (items have to be dragged somewhere else on the application to be edited).

This application demonstrates how a to-do list application does not need to look like a simple, linear list. It also serves as a great example of minimalistic design properties that make the application more manageable and less overwhelming.

2.4 The Hypothesis for a More Useful System

The problem with most to-do lists is that they actually add stress rather than relieve it. “The list usually gets too big and unbearable, or it ends up focusing on the wrong
things first” (Linenberger, 2011, p. 43). By contrast, a better to-do list will allow the user to see what their real priorities are and not feel overwhelmed by lesser priorities. In addition, a to-do list that allows users to measure the time their tasks will take and set only realistic goals for each day will avoid the problem of to-do list overload commonly associated with using just one simple list. The goal of this project is to create a new application for managing tasks that overcomes the limitations of those currently available and leverages the latest in applied psychology.
Chapter 3

Literature Survey

When it is obvious that the goals cannot be reached, don’t adjust the goals, adjust the action steps.
-Confucius

As technology has improved and understanding of how the human brain has advanced, opportunities have been created to improve the tools individuals can use to plan. A survey of the available literature regarding how modern psychology affects human interaction with software applications can suggest how best to design them. Colors can be used to indicate grouping, and the use of specific colors can make differences between colors easier to perceive. In addition, descriptions of human memory and attention suggest better ways to design an interface. Finally, an inventory and analysis of the technology currently available informs design decisions for creating a new planning tool.
3.1 Psychological Background for the User Interface

As Harold Thimbleby writes, “User interface design is a very difficult business. It combines two awkward disciplines: psychology and computer science” (Thimbleby, 1990). Creating a good user interface means designing software in such a way that it leverages what we know about how the human brain works. By applying what is known about how the human brain functions, software engineers and designers can build applications that reduce the cognitive load on their users, thereby making the applications seem easier to use and less obtrusive.

Behavioral psychologist Susan Weinschenk Ph.D. maintains a blog in which she discusses how psychology should influence user interfaces. In her article for UX Magazine, she lists a number of important points to understand when designing user interfaces (Weinschenk (2010)):

1. People want to do the least amount of work possible.

2. Progressive disclosure is the concept that designers should let users choose to get more details.

3. Show examples of how things can be done.

4. Make user interface components look like how they should act (e.g. knobs, sliders, etc.), based on the concept of affordance.

5. Be minimalistic with features to avoid overwhelming the user.

6. Defaults are important to allow users to spend less time configuring the application.

7. Anticipate that users will make mistakes. Make it easy to undo actions.
8. People want to know what’s coming, so give status updates (e.g. “Loading...” or “Deleting...”).

9. Use color to show items that belong together.

By incorporating these attributes into an application design, the developer can not only help the users accomplish their goals for using the application more easily, but also make the application more enjoyable to use since there is less effort required by the user.

3.1.1 Psychology of Visual Perception

German psychologists in the twentieth century produced a set of descriptive principles for how human vision works, called the Gestalt principles. What they found is that the human brain is optimized to see structure. There are numerous such principles that describe how human vision is designed to process the environment, which affects how humans interact with it. Jeff Johnson, Ph.D., has explored the implications of this psychology on user interface design. He is president of the product usability consulting firm UI Wizards, Inc. and author of multiple books on user interface design.

One such principle explained by Johnson describes how the eye automatically separates objects presented over other objects. When one shape is presented within another, the eye tends to interpret the smaller one as the foreground and the rest of the scene as the background. This is useful, since this can be used to grab the user’s attention by presenting important information over the rest of the on-screen information and setting it up to look like it is “over” the other information (Johnson, 2010, p. 20).

In addition, the visual system uses similarity of objects to separate them into groups. Objects and shapes that look similar are perceived as being within a category
together. This means that an application can use similarity in shapes and objects to represent groups to the user. For example, tasks that are the same color will be perceived as belonging to the same group. See Figure 3.1 for an example. In this example, the hollow stars appear grouped (Johnson, 2010, p. 14).

Figure 3.1: The eye perceives similar shapes as belonging to a group.

The human visual system also has idiosyncrasies associated with interpreting colors. One important takeaway from these idiosyncrasies is that humans perceive colors better when the colored shape is large. For example, it is sometimes difficult to discern the color of thin shapes like letters but easier to discern colors of larger shapes. This would suggest that applications avoid colorizing text when there is a better alternative in which the area of color could be larger. For example, it would be better to color the background of a task to indicate its category than to use the color of the text itself (Johnson, 2010, p. 56).

Another peculiarity is how different colors are perceived. Johnson suggests that colors be distinctive and distinguished by saturation, brightness, and hue. The best colors to use in order to draw attention to differences between them are those that require the least amount of visual processing, which are red, green, yellow, blue, black,
and white, as shown in Figure 3.2 (Johnson, 2010, p. 61).

![Colors](image)

Figure 3.2: The most distinguishable colors are those that directly stimulate one of the three color-perception channels and send neutral signals to the other two channels.

Another principle of good design implied by psychology is the reduction of reading required by the application. Providing too much text is likely to be intimidating, making the application less usable. The ideal way to design an application’s instructions is to “use the least amount of text that gets most users to their intended goals” (Johnson, 2010, p. 50).

Another important principle underlying good design is to make messages visible to the user, with proper understanding of how best to do so. The human eye has very little resolution towards the edges of the vision, away from the fovea (See Figure 3.3). This means users are not likely to notice messages in their visual periphery. Therefore, important messages should either be displayed in a prominent position on the page where the user is likely to be looking, or the application should call attention to messages if they are displayed outside of the user’s expected center of vision. Johnson suggests using error symbols and the color red to call attention to an error message (Johnson, 2010, p. 67).

### 3.1.2 Psychology of Memory

Steve Krug is a usability consultant who authored a seminal book on web usability in which he urges, “don’t make me think!” According to Krug, the most significant way to make a site easier to use is to make it effortless on the user’s part (Krug, 2005, p. 11). In his book, Krug explains how users actually view web pages and how to design for this. This is why psychology can provide crucial information to software designers, since this field can help developers understand what comes easily to users
Figure 3.3: The human eye has many photoreceptors in the center, or fovea, of the eye but fewer towards the edges, leading to poor peripheral vision. Subfigure A approximates the resolution of human vision, and Subfigure B simulates how the eye’s ability to distinguish letters decreases as the letters move away from the center of the fovea.

and what requires more mental effort.

Johnson explains how short-term memory constraints should influence interface design. One such suggestion is to avoid the use of modes (e.g. “reorder mode” for when the user wants to reorder tasks, or “edit mode” for then the user wants to edit task titles) in an application, since these require the user to recall the currently active mode before proceeding. For example, the reorder mode in the Ta-da Lists application violates this suggestion, since a user has to change modes to and from this reorder one and remember which he is in before proceeding. This would be even more problematic if the user was expected to remain in a given mode for a long period of time, over which he is likely to forget the current mode (Johnson, 2010, p. 86).

Another related suggestion is to provide instructions in such a way that the user can read them while executing them, without needing to remember them. For example, providing instructions in a dialog box that must be closed before proceeding is bad for usability. It requires that the user remember what was written. It would be better to provide instructions in a way that they can be followed while still visible.
3.1.3 Psychology of Attention

Since the brain is preprogrammed to recognize images quickly, the graphic user interface has rapidly overtaken command-line computer terminals. Requiring users to remember and type commands is much more difficult than just having them recognize and click buttons, menus, and icons. Taking this a step further, Johnson suggests using graphic icons where possible in applications, such as to indicate a warning message or a commonly-used function. This is why, for example, modern operating systems like Mac OS and Microsoft Windows have so many icons (Johnson, 2010, p. 114).

Another general principle espoused by Johnson and supported by Krug is that computer applications should "minimize the amount of attention users must devote to operating them" (Johnson, 2010, p. 131). This seems obvious; wherever the computer can do work for the human, it should. Of course, there is a tradeoff with how much control the user exercises over the program if too much is taken for granted. That is why it is incumbent on the developer to understand his potential users well enough to know what can safely be decided for the users ahead of time and what to let them decide. Mastering this balance is the mark of a great application.

Johnson suggests that developers can reduce the load on the user in a few relatively straightforward ways. First, he suggests that the application indicate its status to the user so he does not have to recall or calculate it; the application should inform the user what it is presently doing. This can be done with a simple status bar informing the user of what has just happened or what is still in the process of happening. Next, the application should never make the user diagnose a problem, such as a poor Internet connection, when the application easily knows or can determine the error itself. This means generic error messages should be avoided where possible. If the application can determine what the problem is, it should do so and suggest ways to solve it. Another way to reduce the user’s cognitive requirements is to decrease the number of options and settings. Designers may think they are empowering users by giving them more
settings to adjust, but in reality these are usually unnecessary and complicate the application. Finally, the developer should make the system seem familiar by using common metaphors. The concept of a desktop on modern operating systems is one such example, as is the concept of divided tabs in a notebook in Microsoft Word for Mac, as shown in Figure 3.4.

![Microsoft Word for Mac using familiar notebook metaphor](image)

Figure 3.4: Microsoft Word for Mac uses a familiar notebook metaphor, including tabs dividing sections.

Another important aspect of a user’s attention has to do with how responsive the application is to the user’s input. Ideally, there would be no delay between when a user performed an action and when the result happened. However, this is not always possible. There may be delays associated with transferring information over the network, writing data to a disk, or processing large quantities of data. Responsiveness, however, is not entirely dependent on how quickly an action is performed but whether the application communicates with, and responds to, the user while executing time-consuming operations. An application that suddenly freezes every time it saves to the server would be unresponsive and frustrating. A better application would inform
the user that it is in the process of saving to the server and, ideally, not prevent the user from continuing to work while this is happening.

To appear responsive to a user, the application needs to maintain the appearance of cause-and-effect and respond smoothly. For example, the application should immediately acknowledge user actions, even if the application cannot immediately satisfy the request. In a web application, this might mean first identifying the user’s action and then confirming when it was saved to the server a few fractions of a second later. Similarly, for an application where there is animation, these animations should appear fluid and smooth, otherwise they distract the user from whatever metaphor the animations were intended to support. For example, in iBooks on the Apple iPad (Figure 3.5), the animation for page turns is remarkably smooth and accurate to the position of the user’s finger. According to Johnson, the term “instantly” means 0.1 second for the brain. After this duration, the brain no longer perceives the action as instantaneous.

Figure 3.5: The page turning animations and shading details are smooth enough to maintain the book metaphor in iBooks on the Apple iPad.
3.1.4 Psychology of Motivation

Princeton Professor or Psychology Daniel Osherson observes that, “It is nearly impossible to motivate anyone to anything. They have to want to do it.” This is a challenge of any to-do list: making the user want to strike items off the list.

Fortunately, among the literature on motivation, there is a general consensus that goal-setting is an essential component of motivation. Therefore, a to-do list that allows users to set goals is an important way of motivating them to actually accomplish them. “Put it in writing,” seems to sing the chorus of productivity experts and psychologists. For that purpose, a list is ideal, because it represents an articulation of these goals.

Another aspect of motivation is not feeling overwhelmed. When an individual looks at his to-do list and sees far more on the list than can be accomplish by when it needs to be done, this is not just overwhelming, but it is also paralyzing. The person doesn’t know where to begin when the list seems so large that it is impossible to accomplish it all. The goal of a system that motivates users to do their tasks should be to ensure that the users are never overwhelmed.

The system needs to give them a sense of control over their situation and make them feel at ease. This can be done by guiding the user in creating goals that are not so large that they seem insurmountable. For example, a good application would make sure the user does not try to accomplish more than is reasonably possible in a day, fail to do so, get discouraged, and become less productive. The application can do this by letting the users set time estimates for their activities, schedule them over the course of a day, and reflect on whether the total time scheduled for that day is reasonable. Amy Lynn Andrews calls this the “secret” to time management. “If you want to manage your time, the sum total hours of your daily activities should be less than twenty-four” (Andrews, 2011, p. 143).
3.2 Technology Available

Before building an application, it is necessary to determine the platform on which the application is to run as well as what technology will be used on this platform. Applications can either be developed to run natively or in a browser, the latter of which is the most accessible option. Developing for a web application, however, requires writing two parts of the application, one part for the server and another for the client. Fortunately, software libraries and frameworks like jQuery and Django make this easier, so the developer can focus on more important aspects of the development, such as the user interface.

3.2.1 Platforms

A major choice when building any application is what platform on which to build it. The platform decision has far-reaching implications. It affects what the final application ultimately looks like to the user, how they interact with it, and how the developer has to design it. It also affects what audience can access the application and where and how they access it. In short, this is likely the most fundamental decision of the entire application.

Native Application (Mac, Windows)

One possibility is to develop natively for Mac or Windows. On the positive side, native applications can offer very well-integrated experiences for users, especially when it comes to user interfaces. While many design idioms (like drag-and-drop) have been replicated using special frameworks for web applications, these implementations are relatively limited when compared to what can be accomplished using the operating system’s built-in utilities. Also, native applications can access the user’s local disk to store and retrieve information. Additionally, native applications are less likely to rely
on, and be slowed down by, communications over the Internet, since the processing is generally done on the user’s local machine. This means applications might be less prone to slowness caused by the network.

Developing natively would involve writing software in languages such as C++, C#, or Objective C, and using integrated development environments (IDEs) such as Microsoft Visual Studio for Windows or Xcode for Mac. These languages are full-featured and afford the developer considerable control over the application, but the tradeoff is between simplicity and portability. These types of applications are complex to build, and, more importantly, they are not very portable. An application designed to run on Windows, for example, will not run on Mac without a complete refactoring of the code.

Once developed, this software needs to be downloaded by the user. Apple has launched an App Store for the Mac OS X operation system for desktops, which streamlines this process of distribution. Windows does not have a similar “App Store” concept, but software can easily be distributed online by download. It seems entirely impractical to even consider distributing software on physical media such as CD-ROM anymore given the proliferation of Internet access, especially broadband Internet access, which make it much cheaper for publishers to distribute applications to consumers.

When considering what platform on which to develop, maintenance is also an important consideration. Seldom are applications built that do not require or lend themselves to updates throughout their lifetime. While there has been an increase in applications that can automatically update themselves with little user intervention, most updating still requires the user to take a positive action to initiate the download or installation of patches or updates. Although the Mac App store provides powerful update capabilities with close connection to the operating system itself, this is still suboptimal when compared to cloud-based applications where updates can be rolled
out instantaneously to all devices.

**Mobile Application**

Over the last few years, mobile devices have become more prevalent and more powerful. Therefore, it seems important to consider native application development on mobile operating systems. The convenience of this approach for the user is that they are likely to have their mobile devices (predominantly smartphones) with them wherever they go, and it is easy to interact with these devices even on the move in situations where a laptop or other larger computer would be inconvenient or impractical. Smartphones, for example, can even be operated using one hand. They are incredibly accessible.

However, there is a tradeoff for this portability and size. Mobile applications are limited by the small screen sizes of the devices. Less information can be displayed on smaller screens, and they provide less room for user interface components such as buttons. This can be incredibly limiting when it is important for the application to present large amounts of information and let the user interact with it. To display this information requires making it smaller, but making it smaller makes it hard to interact with.

Another downside for mobile applications is the variability of the ecosystem. There are dozens of mainstream devices and a handful of platforms on which to develop. The main ones, of course, are iOS for Apple products like the iPhone and iPad, Google’s Android for a variety of devices from multiple manufacturers, Windows Phone OS, and Blackberry. There are few good ways to develop applications that work on multiple platforms. Even large, popular applications like Skype, have trouble rolling out features to their applications across platforms at once. The screen sizes for these devices are different, the capabilities of their operating systems are inconsistent, and the ways in which users interact with the phones vary (from touch screens on
iOS devices to a combinations of keyboard and touch screen on certain Android and Windows phones, to keyboard-and-cursor on some Blackberry models). This poses challenges, especially for applications where the interface is itself a feature – when the platforms do not lend themselves easily (or at all) to certain interface strategies. For example, drag-and-drop does not translate well to Blackberry phones with cursors guided by trackballs.

As for distribution, mobile applications represent the vanguard in software distribution strategies. Desktop operating systems seldom have software to assist users search for and install applications. However, this began in 2007 with the iPhone, whereby mobile applications are primarily distributed by app stores with close ties to the mobile operating system itself. In fact, for iOS, the only way to install an application is using the operating system’s built-in App Store or that on iTunes. It’s not possible to download an application from the Internet in a browser and install it.

App stores, or, more generally, integrated app distribution networks, have made it not just easy to distribute software, but also to extract payment from customers. These networks usually let users purchase applications just by entering a password to log into their account, rather than requiring typing in credit card information repeatedly. In addition, these networks have made paying small sums of money for software more acceptable to users. While these app stores may deflate the prices of software, the larger audiences to which they make these applications available – combined with the ease of paying for applications – increases the volume of paying customers significantly enough to offset the impact of the deflated application prices.

With respect to maintenance, these app stores make it easier for developers to roll out updates to customers and easier for customers to download and install them, abstracting much of the complexity of these updates. Updating applications is as much an aspect of mobile app stores as the initial download by the customer.

Unfortunately, this market is still undergoing rapid updates and changes. Screen
sizes (in real sizes and pixel counts) are changing very rapidly, as are the capabilities and application programming interfaces (APIs) available to developers. Features made available in one release of the mobile operating system might be deprecated shortly thereafter, as happened with Unique Device Identifiers (UDIDs) in iOS. In fact, applications that use UDIDs are now being rejected from the App Store because of privacy concerns, even though this resource was readily accessible in previous versions of the operating systems. Any developers that made use of this API must now modify their applications to remain in the App Store.

**Web application (Mobile)**

A potential workaround to many of the drawbacks of developing for a mobile device is developing a web application for mobile devices. This circumvents the regulations of each device maker’s app stores, as Google did with the Google Voice app in 2009. Since the Google Voice app duplicated some of the core features of the iPhone, namely, making and receiving calls, Apple, according to its policy, rejected this application from the App Store. Therefore, Google made a web application so that users could access the application without needing to install an application. All they had to do was visit the URL of the web application. Apple could not (or at least, in this case, did not) prevent users from accessing a website.

Designing a web application also has the benefit that it can be, by default, multi-platform compatible. Since most web standards are supported by all browsers, regardless of device, it is relatively straightforward to develop an application that can be accessed from any device with internet capability, even ones yet to be conceived.

There are, of course, challenges here, too. For one thing, not all devices are touch screens, although this is increasingly becoming the norm. That means that these web applications, depending on the devices for which they are intended, might not be able to employ touch events (as opposed to conventional click events for desktop
computers) or might need to have multiple interfaces for the same application to be presented depending on the device accessing it.

The main downside, however, is that mobile web applications can be rather slow. First, they rarely can take advantage of the full resources of the device, since they depend on the speed of JavaScript and the browser’s rendering engine, which can be rather heavy and suboptimal. Second, these applications are more likely to rely on exchanging data between the client and the server, which – especially if this takes place over a cell tower connection rather than WiFi – could be rather slow and involve high network latency. Combined, these are serious drawbacks to this approach.

**Web application (Desktop)**

What is available on the internet has undergone significant changes in recent years as a result of two main developments: availability of low-cost, low-effort hosting and open source frameworks that make web development more accessible. Combined, these developments have made it easy and inexpensive to build web applications that rival or surpass their desktop equivalents. The advancements here have even led to significant developments using these tools by large companies for whom complexity and cost are less of an issue. Google’s GMail, for example, is a full-featured email client that rivals – if not trumps – the mainstream desktop equivalents such as Mac Mail and Microsoft Outlook.

The advantages of GMail being a web application are numerous. First, customers can access the application from just about any computer with internet access, not just their own, meaning the application is infinitely portable. The application doesn’t need to be preinstalled on the system for them to use it. The code for GMail is loaded anew each time the user visits the web site (technically, it may be cached, but these caches can be invalidated by GMail when the code has changed, forcing an update), which means that Google can easily launch updates to the software that will impact
all of the users at once. There is no need to install updates or patches. This means that web applications often have rapid design cycles – publishers can frequently roll out updates to the software. It could also be argued that web applications can be even more secure than native applications because publishers can immediately release updates to the application to address security vulnerabilities, while publishers for native applications must get the end users to download and install the updates.

Not only does this make it easier to distribute updates, but it also allows the software publisher to test features on users. The publisher can ultimately control what version of the software is sent to the client to load, and this means the publisher can select a subset of users to sample new features and ask them for feedback before rolling this out to other users. While this is technically possible with other distribution media for native applications, it is remarkably easy by comparison to do so with a web application. The result is that users of web applications potentially get to enjoy new features sooner than would counterparts using native applications, since a publisher would usually wait to release updates to the software until it had a significant number of new features or bug fixes. In some cases, new features might not be added until an entirely new release of the software, which in the case of Microsoft Office, might take years. Web applications, therefore, have an advantage of being able to more easily be on the cutting edge.

With recent advances in HTML, especially HTML5, and increases in speed of JavaScript processing within browsers, GMail has been able to make use of cutting-edge technology to create an experience within a browser that matches or exceeds that of native applications. One such technology that GMail pioneered is Ajax, through which the server and client can exchange data without needing to reload the page. The greatest drawback for GMail as a web application is that the data must constantly be exchanged with the server, which can add delays to the application and decrease its responsiveness. GMail uses caching strategies to minimize this, but it is still a
factor when the user’s actions outpace the cache or when the network connection is slow.

Perhaps one of the most significant benefits to the developer of writing a web application is the unsurpassable ease of “distributing” the software. The software does not need to be downloaded on the client’s machine in the typical sense. The software publisher just needs to serve the web application from a server. Published once for theoretically limitless users (where the limit is generally set only by server capabilities), this software is highly available. There are no app stores to deal with, nor does it require a major commitment on the user to download and then install the software. Web applications don’t need to be installed, so they are much easier to access.

3.2.2 Choosing a Platform

There is no clearly better platform on which to develop. Instead, it depends on the software publisher’s priorities and the needs of the application. For example, it would be rather difficult to create a high-definition video editing application in the browser because such an application would need access to files on the user’s computer, which is not easy, and it would either necessitate transferring huge video files over the Internet, or require calculations to be done in the browser in JavaScript, which would be much slower compared to compiled code running a native application. Developing a web application seems ideal when the software does not require access to the user’s files, when the computation required is not extensive, and when the data that would need to be exchanged is minimal. Developing a native application seems ideal in the opposite cases above and also when sophisticated animations or complex user interactions are required, for which a browser would come up short.

In weighing the options, it seems that a web application is the ideal platform for a to-do list. The application can be highly available (accessed from any Internet-
connected device), will be multi-OS friendly (not restricted to Mac or Windows), does
not require exchanging large amounts of data (just the to-do items themselves and
perhaps user preferences), and allows the publisher to roll out updates and patches
frequently.

Creating a web application actually involves writing two separate codebases – one
application runs on the server and one application runs on the client. The client
software allows the user to interact with the data – it provides the input/output
interfaces. It also negotiates data transfers with the server. The software on the server
stores and keeps organized this data, and provides it to the client upon request. These
two components of the overall application are designed using different technologies,
and they communicate with each other through standards-compliant protocols and
formats.

3.2.3 Client-side Technologies

A web application is one that is access through a browser. There are a number of
ways to create web applications. For example, developers can use Adobe Flash to
create applications with very rich interfaces, or they can develop applications using
HTML and JavaScript. Since HTML only provides capability of presenting data in a
given state, these technologies are needed to allow the user to interact with the page
and manipulate the state of the page once it has been loaded.

Flash

Adobe Flash (formerly Macromedia Flash or Shockwave Flash) is a platform for
interactivity on the web. It is a programmed platform, with ActionScript being
one such programming language that can be used to manipulate the animations and
interface. While Flash gives the developer significant control over the user experience
and enables them to present impressive-looking interfaces, it also has some significant
downsides.

In fact, Steve Jobs, former CEO of Apple, Inc., wrote a scathing note in April 2010 explaining why Apple was not going to support Adobe Flash on the iPad. Although his article is a defensive reply to Adobe’s claim that Apple was depriving its users of a full experience on the web, and therefore cannot be relied on as an objective review of Flash, Jobs’ article does enumerate many of the objective downsides of Flash (Jobs, 2012).

1. It is a proprietary platform that does not support open standards like HTML5, CSS, and JavaScript.

2. It is prone to crashing, security vulnerabilities, and performance drawbacks.

3. Video played in Flash typically is not optimized to use the H.264 decoder built into many chips, and is therefore less efficient.

4. Flash does not operate well when the user interface is touch, as it is on mobile devices like iPhones and iPads.

Even web browser publishers are taking note. It is generally understood that “Flash and other plugins are often responsible for poor performance and security vulnerabilities” (Gilbertson, 2012). In order to alleviate these problems, Mozilla is considering adding a feature to the popular Firefox browser that requires users to click on plugins on a web page to let them play. Jared Wein, a developer on the Firefox team, considers this “an incremental step towards securing our users, reducing memory usage, and opening up the web” (Wein, 2012).

**JavaScript**

JavaScript is the main way developers can build interactivity into their websites and web applications. JavaScript enables developers to set up event-based scripts (e.g.
do something when a user clicks a button) and even exchange data with the server without reloading the page. An important concept used by JavaScript is that of the Document Object Model (DOM) in which the page is represented as a tree with nodes, each of which can be manipulated using JavaScript.

There are a number of benefits of using JavaScript. Primarily, JavaScript is implemented by all mainstream browsers and, unlike flash, does not require downloading and installing a plugin. All browsers come ready to handle JavaScript. Not only that, but modern browsers have been significantly increasing the speed at which they can interpret and process JavaScript, no doubt in response to the increasing number of applications that rely heavily on JavaScript (such as GMail). Another important benefit for the developer is the vibrant community of web developers using JavaScript. Because JavaScript is so popular among web developers, there are many web pages online with developers discussing how they handled various challenges and even providing sample sections of code. Furthermore, there are a number of special software libraries that make JavaScript even more powerful.

### 3.2.4 Server-side Technologies

The server is a vital component of a web application, since it is the source of the code that is transferred to the client to be executed to produce the user interface and the data to support this interface. While the client program directly interacts with the user, the server only interacts indirectly with the user through the ways the developer has designed the combined system of server and client to act.

The server technology, therefore, has little direct impact on the user, aside from performance issues. That is to say that the choice of technology on the server side does not depend very much on information about the user. For example, if the user is accessing the application over the Internet, it doesn’t matter whether he is accessing the application from an Apple or a Microsoft computer, from a mobile device like an
iPhone, or from a full computer like a desktop. These devices all communicate over the Internet in the same standard fashion through protocols, such as the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

Therefore, the choice of server technology is really driven by concerns about costs, reliability, and ease of use (perhaps also, for certain requirements, performance). Over the last decade, the cost and complication for setting up web servers has decreased considerably, especially with cloud services like Amazon Web Services (AWS). AWS offers a pay-as-you-go platform for web services such as databases, storage, and computing. These resources can be used for hosting enterprise applications, web applications, media sharing, and even high-performance computing. It is no longer necessary, nor even preferable in many cases, to set up a physical server in one’s home or office when the same can be achieved with no direct overhead by paying either for a small slice of a massive virtual server (such as with Amazon) or even just renting a dedicated server from a provider such as GoDaddy.

Because they are free and there is a vibrant supportive community, Unix-based operating systems have become very popular among developers who do not have significant financial resources to purchase proprietary systems. This has led to a snowball effect where more and more free software is generated to empower developers to create more advanced systems than would have been possible without these tools. For example, the Python language underpins the Django web framework, which is a remarkably simple yet robust platform for developing a server-side application. The power comes from the layers of abstraction that allow developers to create sophisticated web applications without having to manage aspects of the development that are not value-added. In other words, these underlying frameworks allow developers to focus on higher-level development concepts, like making the application look and feel great, instead of focusing on doing the setup work. These frameworks automatically take care of much of this necessary work. The result is that developers can spend
more time focusing on the user interface than on the intricacies of the server.
Chapter 4

Market Segments

You can’t just ask customers what they want and then try to give that to them. By the time you get it built, they’ll want something new.

-Steve Jobs

Before developers can hope to solve the problems their prospective users have, they need to begin by identifying what these problems are. This application has implications for a variety of market segments, especially corporate professionals, academics, and individuals managing personal tasks. When there are multiple categories of users that will be using the application, it is good practice to consider each of them separately to understand which needs are unique to all users and which are common to them all. Based on this information, the developer can then decide how to prioritize the features within the application.

4.1 Various Segments and Their Needs

To-do lists can be useful to many different types of people, all of whom have one thing in common: a need to get things done. By identifying individual segments of users, a developer can better understand what these users are trying to accomplish
and how best to help them do it.

The way to properly design an application for these market segments is to create a persona for each, as Alan Cooper suggests. Cooper is a user interface design consultant and the creator of Visual Basic, a popular programming language and development environment for Microsoft Windows. He suggests that developers should “make up pretend users and design for them” (Cooper, 2004, ch. 9). By constructing these archetypes of users, the developer can then more effectively describe the needs and capabilities of users. As user experiences specialist Robert Hoekman concurs, “personas are also meant to help those involved with a project realize that the person on the other side of the screen is not made of rubber, and cannot be bent at random to fit arbitrary ideas of what a user can handle and will find useful” (Hoekman, 2010, ch. 3). This can go even so far as naming these personas and including photos to represent them. This strategy of developing personas is a crucial part of how divisions within Microsoft design software, testifying to its impact on the software development community.

It is best to approach developing an application in this way because “a user’s goals live outside the application. Their goals are personal” (Hoekman, 2010, ch. 3). Someone using the web application does not really care much about how sophisticated the algorithms are that make it work. The user just wants to find out how many cups are in a quart, what the company’s income was in the last quarter, or how best to manage their time.

### 4.2 Corporate Professionals

The first persona is that of the corporate professional. These individuals represent a significant market share and are willing to pay for software tools that will make them more productive, making this segment an important one to consider.
4.2.1 Persona: Brad

Brad is a 32-year-old corporate professional who needs to manage his time. 
Source: http://www.icis.com/

Brad is an 32-year-old executive at Initech. He frequently feels overwhelmed by 
his responsibilities and is in need of a tool to help him organize his responsibilities. In addition, he needs to manage a variety of projects at once, keep track of important 
deadlines, and prioritize his workload. Brad is typically on-the-go, traveling to clients around the world. He is familiar with Microsoft Outlook and Excel and has an iPad on which he reads eBooks.

4.2.2 Software Needs

Because Brad is always on-the-go, he cares that he can access his to-do list from anywhere his work takes him. This is why Brad would benefit from a web application that makes it easy to access the application from anywhere. This means Brad could log on and modify his to-do list from his own laptop, a hotel terminal, or even a colleague’s computer, regardless of the operating system or browser used on the computer. This is why installed software (i.e. native applications) would not work
well for Brad and why a web application with strong compatibility and accessibility is preferable.

While Brad is an educated person and could use more advanced features, he is generally uninterested in spending much time organizing. Therefore, Brad’s ideal application would take advantage of common metaphors with which he is accustomed that would allow him to modify his tasks easily and read and understand his lists without much effort.

Since Brad may want to store important deadlines in this application, losing his data would be a costly and very frustrating problem. That is why Brad values having peace of mind that his data is protected against loss. He does not care very much how it is protected, so long as it is protected. For example, Brad would be willing to download an XML file to back up his tasks. However, since Brad is busy as it is, it would be better for the application itself to manage backups. Perhaps the application could email Brad a backup copy of his tasks each day or each week.

Lastly, since Initech has many competitors who could seriously compromise the company’s interests, Brad needs to be confident that his information is secure from prying eyes. That is why Brad would strongly prefer an application that used secure socket layers (SSL) to encrypt the traffic between his computer and the web server. Additionally, Brad may want (and be willing to pay for) even stricter security, such as encryption on the server itself.

4.3 Academics

Professors and students engaged in academic pursuits similarly use tools to organize their projects. Professors have to manage classes, research, and administrative issues. Similarly, students manage responsibilities ranging from classes to extracurricular activities. Students in particular, who have grown up using software to organize their
lives, constitute a significant market segment.

4.3.1 Persona: Jeff

Jeff is a 21-year-old student at Princeton who needs to manage his assignments and extracurricular responsibilities.
Source: http://www.brandsizzle.com/

Jeff is a 21-year-old Princeton junior in the Department of Operations Research and Financial Engineering. He is enrolled in five classes, a member of an a cappella group, and on the varsity hockey team. He wants an application to help him keep track of his readings, assignments, and other responsibilities. Jeff is very tech-savvy and organized; he uses an iPhone so he always has his calendar and email handy, and he likes to spend a few minutes each morning planning his day.

4.3.2 Software Needs

Since Jeff has many different categories of responsibilities (classes, extracurriculars, personal, etc.), he does not want to confuse them, so for him it is important to be able to separate his tasks by these categories. For example, he might want to be able to identify all of the tasks for a specific class.
Additionally, Jeff often needs to undertake large projects with a variety of deadlines. Therefore, he needs to be able to break these larger projects down into smaller, more explicit, and attainable goals. He also needs to be able to differentiate easily between when something is actually due and when he plans to work on it. For example, for an essay due on Friday, Jeff may plan to work on it on Monday so that he can have a draft to revise by Wednesday.

While Jeff prefers to organize his tasks according to his own way of scheduling them, he has little control over when many of his assignments are due. That is up to the professors that assign them. They do so by posting the assignments to Blackboard, a software platform for education. It would make Jeff’s work considerably easier if this application could easily download the tasks, such as assigned reading, from a data feed provided by the professor’s syllabus via Blackboard. Jeff would still want to control when he actually does these activities, but the due dates for these assignments are already set by the professor. If the professor updates the due dates for certain assignments, the application could even alert Jeff to the new deadline and automatically update his task list to reflect the changes.

Jeff is not just responsible to professors but also to his hockey coach. Jeff’s coach has a fitness program that includes daily workouts Jeff is expected to complete. Like how professor’s assignments could be downloaded to Jeff’s to-do list, it would make his life easier if these workouts could also be downloaded automatically from a feed provided by the coach. It would be even more useful to Jeff if the application would notify his coach when he completed a workout task. Since Jeff is likely to need this information at the gym where he has his iPhone, it would be important that Jeff could view and update his tasks on his mobile device.

Since Jeff frequently needs to cooperate with classmates on projects, he would like to be able to use his task list to help him coordinate. In a class for which he has a group project, he would like to be able to share tasks with other members of the
group. For example, all members of the group may need to send him their section of
the project by a certain date. To facilitate this, Jeff would like to be able to share
tasks with them easily.

Since Jeff likes to plan his days rather methodically given how busy he is, he would
like to be able to easily view his list of tasks for the day. This means the application
he uses should make it easy for him to print out a list for the day. The application
could even send Jeff a reminder email each morning containing a list of tasks for that
day.

4.4 Around the House

To-do lists can be useful to individuals outside the boardroom and the classroom.
Individuals, regardless of occupation, have personal tasks to manage. Although these
tasks may seem less critical than those at work or at school, they can be organized
nonetheless and are sometimes just as important.

4.4.1 Persona: Molly

Molly is a 38-year-old mother of two. She works part-time as a yoga instructor and is
active in her church community. Molly wants a tool to help her remember all of the
things she needs to do each week, from doctor’s appointments for her children to yoga
sessions. She currently uses a piece of paper clipped by magnet onto the refrigerator
in the kitchen. She uses her computer to browse the web, check email using AOL,
and make flyers in Microsoft Word.

4.4.2 Software Needs

Molly’s main concern is simplicity. She doesn’t want to spend time organizing her
list each day or week; she just wants to jot things down and not forget them. For her,
Molly is a 38-year-old mother of three who needs to manage a variety of tasks. Source: http://www.sheknows.com/

it’s important that she can navigate quickly between the weeks ahead and add new tasks very quickly. She doesn’t want to sit around and assign priorities to her tasks. If it is on her list, she considers it a top priority.

Since Molly is used to using pen and paper and has only basic experience using computers, she will need the program to be as self-explanatory as possible. Affordance throughout the application will be critical so she can understand how to use it without needing a manual. Things that can be dragged should look draggable, and things that should be clicked should look clickable. For example, the application should communicate well with the user to guide her through her basic tasks. The purpose of clickable icons should be obvious, or there should be helpful text within the application to explain them. This is especially critical during the first few times she uses the application. If there is an online help forum (which she is unlikely to use) it should be basic, not advanced. It should cover the main actions she would be interested in.

As a convert from using a pen and paper to organize her tasks, Molly is used to having a physical sheet always available showing her what to do for the day. That’s why Molly would like the application to provide a printable copy of her weekly tasks.
She would want to then put this on the refrigerator or somewhere in the kitchen so she can have it handy. She might even want to fold it up and carry it around with her throughout the day.

Molly’s home Internet connection is sometimes slow, and she often gets frustrated if the application she is using is moving slowly and not responding. Therefore, the application should provide her with status updated as it is sending data back to the server. However, this should all happen in the background. The application should not freeze while it is sending data to the server, as this would dissuade Molly from making the switch from just writing it down. Instead, it should indicate what it is doing in the background and let her continue working.
Chapter 5

Using the Application

Just remember: you’re not a “dummy,” no matter what those computer books claim. The real dummies are the people who – though technically expert – couldn’t design hardware and software that’s usable by normal consumers if their lives depended upon it.

-Walt Mossberg

It is difficult—perhaps impossible—to design an application that perfectly meets the needs of all its users, but it is a worthwhile pursuit. While this application does not address all of the needs of the identified market segments, it addresses a number of the most significant ones, and it is designed based on the suggestions of modern psychology.

5.1 Application Overview

The presentation of the application is important because the user relies on what he perceives to understand how to use it. He immediately forms impressions about how to use the application based on what the application looks like. Figure 5.1 shows an example of how the application may appear.
Figure 5.1: The application as a user may see it. The tasks appear in different colors according to the projects with which they are associated.
5.1.1 The Calendar

Figure 5.2: The calendar section of the application is the main focus of the application. The current day is highlighted in blue.

Upon first loading the application, the user sees a calendar presented with the current week. The current day of the week is highlighted in a light shade of blue to stand out. The background of the calendar is a grid, which suggests that tasks will appear in this grid and size will be an important characteristic.

There are various arrows to the sides of the calendar, as shown in Figure 5.2. When the user moves his mouse over these icons, they light up and the cursor changes from an arrow to a pointer, indicating that these are clickable. When a user clicks these arrows, the calendar slides to reveal the days in that direction. The number of arrows in each icon indicates a relative amount of time to advance the calendar. One arrow advances the calendar by one day, two arrows by one week, and three arrows by one month. The user can repeatedly move the calendar in any direction, as this calendar appears to span infinitely in either direction, out of view.

In addition, if the current day is not visible on the calendar, a rectangular “Today” icon appears on the side of the calendar where today would be located offscreen. When the user clicks this icon, the calendar automatically slides to reveal the current date.
Figure 5.3: The project bar provides the user with additional flexibility for managing complicated projects or categories of tasks. The user can pick colors for each project, which are represented by the thin colored bar below each title.

5.1.2 The Projects Bar

Below this calendar is a section that appears similar, but instead of dates, it lists headings. The default headings are “Personal” and “Work.” This suggests that the user can use these columns to create special categories of tasks or projects. The user, however, can customize them, as shown in Figure 5.3.

In addition to these different titles, each heading also includes a thin rectangles of a certain color. This indicates that there are colors associated with the tasks. Further, when the user’s mouse moves over these headings, a paint bucket appears in the lower-right corner of each heading. The user’s mouse turns into a cursor when it is over these icons, suggesting the the user can click these icons to manage the color of these categories.

When a user’s mouse hovers over the heading of one of these categories, what appears to be a gripping strip (e.g. something rough, dotted like sandpaper) appears in the upper-left corner. Upon further inspection, the cursor changes to a multi-directional arrow to indicate that these categories can be moved.
5.1.3 The Top Bar

At the very top of the application, as shown in Figure 5.4, the user sees the title of the application, the name of the list, and two links for adjusting settings and logging out. The name of the list has a down arrow next to it. When the user’s mouse moves over the list name or this image, this arrow glows brighter and the mouse changes to a pointer icon, hinting that the user can click this to reveal more options.

Directly below this is a textfield labeled “Quick Add” with placeholder text in it that reads “e.g. 'Call Jeff @Monday.'” This indicates that a user can add tasks by entering them in this box. It even describes the proper format for adding tasks, where the user can enter the date after the @ symbol.

5.2 Adding Tasks

The most common action a user is likely to take is adding tasks to the list. That means that this action needs to be simple. This is, in some sense, the first impression the user gets of the application. There are three ways of adding tasks: double-clicking, dragging for the desired size, and using the quick add tool.
5.2.1 Click to Add

The first way of adding tasks is to double-click on a day or project where the task is to go. After this, a new task appears on that day or in that project. This new task automatically defaults to the minimum duration, fifteen minutes. If there are other tasks already there, this new one goes to the bottom of the list. When the task is added, the application’s focus is automatically moved to the textfield for naming the task. This means the user can double-click and then immediately begin typing to name the task; he does not need to click on the task after creating it to activate it for typing. To save what he has typed, the user can hit the enter key, click somewhere else in the application other than the title, or wait two seconds. If any of these occur, the system automatically saves what the user has entered.

5.2.2 Drag to Add

A second method of adding tasks is a slight variation on the first, since it also allows the user to specify the duration of the task while creating it. The user can click down and then drag within either a calendar day or project to create a task of any size. As the user is holding down the mouse button and moving down, the task changes in size accordingly. Once the user has dragged the task to the right size, he can stop holding down the mouse button and the new task will be created. Again, after this, the focus is automatically brought to the title of this new task so the user can begin typing immediately.

5.2.3 Quick Add Tool

These two methods of adding tasks are ideal when the user is already at the point in the calendar where he wants to add the tasks. They are quick and simple – the user just has to point and click to create tasks. However, these are only easy when
Figure 5.5: The quick add bar gives users an easy way to add tasks by typing them in and hitting enter. It features a syntactic analyzer to parse dates.

The user wants to add the task to a day that is already visible. Otherwise, the user needs to move the calendar view to a different week before adding a task, which could be time consuming. To address this, the application provides another way of adding tasks to any date.

The third way of adding tasks is through the powerful quick add bar at the top of the screen, as shown in Figure 5.5. This text field is always available at the top of the window for jotting down quick tasks. Users simply enter the task name and the date and hit enter. The system then parses what they wrote to identify the title of the task and the date they plan to do it. The user has considerable freedom for how he can specify the date, because the application uses an advanced syntactic analyzer. For example, all of the following are valid ways of specifying that a task should be added on Tuesday, April 17th if the current date is Monday, April 16th:

- tues
- tuesday
- tomorrow
- apr 17
5.3 Managing Tasks

One of this application’s main benefits to users is the way it allows them to organize their tasks, even after they have added them. This ease of reorganizing or editing tasks is partly what differentiates electronic to-do lists from their paper counterparts. There are a number of possible changes that users can make, which are listed below.

5.3.1 Reordering Tasks and Moving Tasks to a Different Date

The most significant difference between this to-do list and many others is the way in which tasks can be reordered and moved from day-to-day. Each task can be dragged to a different location, such as a different date, a different project, or a different location within the current date or project. For example, if a user wants to change the day when he plans to complete an action, he just clicks on the task and drags it to the new date, then drops it. Or, if the user wants to reorder tasks within a given date or project, he can just drag the task above or below the others in the same date.

There are a number of visual effects to give tasks the appearance that they can be moved. The shadow effect on the individual tasks implies that these items are not part of the background but that they are slightly elevated above the page. When the user’s mouse is over a task, it reveals a gripping pad, or “handle,” on the left side, which indicates that this item can be dragged by clicking that part of it. Further, when the mouse is directly over this handle, the cursor also changes to indicate that the task can be moved. All of these characteristics lead the user to the conclusion that the task can be dragged.

After the user clicks and drags the task, the application continues to provide the
(a) As the user drags a task to a new location, a placeholder appears indicating where it will end up if the mouse button is released and the task is dropped.

(b) When the user releases the task, it slides into its new location.

Figure 5.6: Users can drag tasks from one location and drop them in a new location.
user with feedback. As the user moves the task around the application, a placeholder appears underneath it in the location where the task will go when it is dropped (See Figure 5.6). Placeholders are the same size of the task being moved and a lighter shade of the original task’s color. Through this mechanism, the user can preview where the task will go before dropping it. After the user releases the mouse to drop it, the task slides from its position to that of the placeholder. It then appears to lower back onto the page to its final, resting position.

This dragging and dropping is effective because it is a familiar metaphor. In much the same way as Apple’s iBooks application replicates the look and feel of a book on an iPad, this application replicates a personal organizer with scraps of paper on it. Just like an individual can pick up and move Post-it Notes around, a TDLoo user can pick up and drag a scrap of paper to another section of the virtual organizer.

5.3.2 Changing Task Durations

Figure 5.7: As the mouse hovers over the task, a handle appears at the bottom which can be dragged up and down to change the duration of the task.

Modifying the duration of a task is intuitive. When the user’s mouse is over a task, the task automatically reveals a handle at the bottom of the task, as shown in Figure 5.7. When over this handle, the mouse’s cursor changes to a north-south resize icon, indicating that the user can click and drag this icon to resize the task.

Resizing is confined to fifteen-minute increments, visualized by the horizontal gridlines on the calendar, the smallest duration of task that can be represented on the list. As the user’s mouse moves up and down while dragging the task, it updates to show the new duration. Even the indicator in the top right of the task, which prints
Figure 5.8: The sizes of the task correspond to the different times the user estimates they will take. By indicating the expected duration prominently, the application can help users avoid overcommitting themselves in a given day.

the duration in words, updates to let the user preview the new duration before “dropping” the resize handle. These different sizes demonstrate graphically the differences in time of each expected task, as shown in Figure 5.8.

5.3.3 Renaming Tasks

Renaming tasks is a direct process. If the task has no name, the user can click where the placeholder text (“Enter a task”) is and begin typing. If there is already a title, the user can directly highlight this text and change it. After hitting enter, clicking another part of the application, or waiting two seconds, the text is automatically saved to the server.

5.3.4 Completing Tasks

When the user would like to complete a task, he clicks the checkbox in the top-right corner of the task he would like to complete. As shown in Figure 5.9, this checkbox is only visible when the mouse is over the task (otherwise, it is a textual representation
of the duration of the task). Clicking this icon initiates an animation that removes
the task from the screen. Within a fraction of a second, the task transitions to
gray, becomes transparent, and shrinks in height. This animation is designed to be
satisfying to make the user enjoy completing tasks on the application.

![Image](image.png)

(a) Typically, a task simply displays the title and duration.

(b) When the mouse moves over the task, it reveals a handle on the left side for dragging the task, a handle at the bottom of the task for increasing its duration, a button in the top right for completing the task, and an icon in the bottom right which brings up a context menu.

Figure 5.9: The application makes significant use of mouseover properties, changing the appearance of user interface items to indicate that the mouse is hovering over it. Hovering over an item can reveal functionality that is otherwise hidden.

5.4 Managing Projects

The project bar at the bottom of the screen provides an additional dimension of
flexibility for the user. These extra columns allow the user to plan the steps of a larger
project and then assign these tasks to days of the week, or just keep a running list of
items not assigned to days. In fact, the user can ignore the calendar section entirely
and use only the project section to manage tasks. This section of the application is
made visible by default, but the user can either toggle this section on and off or change
its height by dragging the separator between the projects and calendar sections.
5.4.1 Showing, Adjusting, and Hiding the Project Bar

A consequence of using different sizes to indicate task durations is that the application requires considerable space to display properly. For this reason, the application gives the user the ability to control how much of the screen is devoted to the project bar. There is a handle in the center of the divider between the calendar and project sections. The user can drag this handle either up or down to change the appearance, as shown in Figure 5.10a. If the user drags this within a certain threshold of the bottom, the application automatically interprets this as an attempt to hide this part automatically. In addition, the user can simply toggle the project bar open and closed using the black tab at the bottom of the window, as shown in Figure 5.10b.

5.4.2 Adding Projects

Users can add projects in much the same way they can add tasks, by double-clicking an empty project column. The new project will be assigned a default color automatically (one that is not already in use, if possible), and the user will be able to give this project a name.

5.4.3 Editing Project Names

One way to contribute to the overall usability of the application is to repeat ways of doing things. For example, editing project titles is done like editing task titles. The user need only click and highlight the text and then begin typing. After hitting the enter key, clicking another part of the application, or waiting for two seconds, this modification is automatically sent to the server.
(a) The user can minimize the projects bar by clicking the project tab at the bottom or by dragging it from the middle. They can use the draggable handle to customize the height of this bar. Users can even expand the projects bar to fill the entire window, hiding the calendar, in order to focus on managing projects.

(b) The user can toggle the projects bar by clicking the black “Projects” tab at the bottom of the screen.

Figure 5.10: Users can show, hide, and adjust the projects bar.
Figure 5.11: The color palette provides an easy way for users to preview colors before selecting them.
5.4.4 Changing Project Colors

The application uses color of tasks to indicate special groupings, based on the Gestalt principle of perception of color. Changing colors assigned to projects, therefore, is an important part of the application. Users can change the colors easily using the paint bucket icon displayed in the heading of each project column. While it is usually invisible, when the user’s mouse enters the row of project titles, paint bucket icons appear next to all active projects. When the user clicks this icon, a small popup appears with color swatches, as shown in Figure 5.11. The user gets a preview of how each color will look when applied when the mouse moves over a swatch. As the mouse moves over each swatch, all of the associated tasks in the project that are visible on the screen temporarily change their color to that of the swatch under the mouse. Finally, when the user has found an appropriate color, he clicks the corresponding swatch to save the change. The colors on the swatches have been chosen according to the principles of vision that suggest that certain categories of colors are easier to distinguish than others, as shown in Figure 5.12.

![Figure 5.12: The color swatches are selected based on the psychology of color perception to be easily distinguishable.](image)

5.4.5 Reordering Projects

Users can reorder projects as they appear in the project bar at the bottom of the screen to better suit their preferences. Reordering projects uses the same drag-and-
drop metaphor as that of the tasks. As the user’s mouse moves over the heading of a project, a handle icon appears in the upper-left corner of the project title, as shown in Figure 5.13. This is the same handle icon that appears on the individual tasks indicating that they can be moved. The user can click and drag the heading to the left or right and drop it into the desired position. After this, the columns are adjusted to reflect the change.

![Figure 5.13](image)

Figure 5.13: The headings of the projects include a handle on the top left for re-ordering the projects, an icon in the top right for deleting the project, an icon in the bottom right for changing the color of the project, and a thin sliver at the bottom showing the color of the project.

### 5.4.6 Deleting Projects

Deleting projects is accomplished much like completing tasks. There is a red delete icon that appears in the upper right corner of the project heading when the user’s mouse is over the heading. This is similar to the green checkmark icon that appears in the top right corner of tasks to complete them. Just as clicking an icon in the top right corner makes tasks “go away,” so does clicking an icon in the top right corner of the project heading.

Deleting projects poses a problem, though, as the application could handle tasks belonging to a project about to be deleted in multiple ways. One such way is to simply delete all tasks associated with a project. Another way is to leave the tasks behind and simply remove their association with a project. Both of these could be reasonable actions, and it might be frustrating to a user if the application acted in opposition to his expectations. On one hand, it might leave behind tasks the user
intended to remove, requiring the user to individually remove them. On the other hand, it might delete tasks the user wanted to keep. Therefore, it seems in this case that the best design decision is to let the user choose how to proceed if there are tasks remaining in the project.

If the user clicks the delete icon on a project and there are no tasks associated with the project, then the application can safely remove the project without needing help from the user. However, if there are still tasks associated with that project, it prompts the user with the two option choices. The dialog prompts the user with: “You are about to delete a project that still has X tasks in it.” It then provides two options: “You can: ‘Leave the tasks’ or ‘Delete all X tasks.’”

5.4.7 Managing Tasks in Projects

A crucial part of the application is the ability to manage tasks within projects. For example, a student working on a long term paper may break this assignment down into multiple tasks (e.g. “Check out books at library” and “Write Chapters 1 and 2”). The application makes it easy to plan a project by laying out the action-steps required first, and then figuring out which days to assign these tasks to. Users can add tasks to projects in the same way they add them to the calendar section: double-clicking or clicking and dragging inside the project. Tasks created within a project are assigned the color corresponding to the project. This is based on the psychological principle that human vision tends to associate like-colored objects as related. Therefore, all of the tasks of a certain color that belong to a project will appear related.

The task retains this project affiliation even when the user moves it to the calendar. When the user moves a task from a project to a day in the calendar, the task retains the associated color. On any given day of the week, the user might have tasks from a variety of different projects. For example, a student might separate each class’ assignments within a project for that class, and a student would probably want to
make progress on assignments in a variety of classes on a given day.

If, however, a task of one project is moved to a different project, the task’s project affiliation and color change to match the new project. Even tasks that did not previously have a project assigned to them that are moved to a project are assigned to the color of that project. For example, if a user created a task in one category, such as “Personal” and then moved it to the “Work” category, the task would thereafter be associated with the “Work” category, including bearing the color of this new category.

Another way to change the project of a task is to do so using the context menu. The user activates the context menu by clicking the yellow triangle with the letter ‘i’ in it. This menu includes an option for changing the task’s assigned project. This is also the way a user can dissociate a task from a project entirely.

5.5 Motivation Features

Having users manage their tasks is just part of what the application should do. The ultimate goal is for the application to help users actually complete tasks. That is why the application should provide tools to help users remember and complete their tasks, such as reminder emails and updates on their progress. Although the features described below are not currently part of the application, the application was designed such that these can easily be added.

5.5.1 Reminder Emails

One crucial way to help users accomplish their tasks is to send reminder emails. The application can support two different types of reminder emails.

The first type is the daily reminder email. This email can be scheduled by the user to arrive in his inbox each morning with a list of tasks, since many individuals check their email when they begin their day. This reminder email will help the user
know what they are scheduled to accomplish.

The second type of email is a reminder email scheduled to be sent at a particular time. For example, if the user indicates that there is a time associated with a task, he can also request that the system send a reminder email a few minutes or hours ahead of time to remind him.

5.5.2 Progress Reports

Another way to motivate users to accomplish their tasks is to track their past accomplishments (or lack thereof). The application can keep track of the number of hours of tasks users accomplished in a day and, if configured to do so, provide this information to the user in the next morning’s daily reminder email. Seeing this past progress could be motivating. On the other hand, the application could also keep track of the the tasks that went uncompleted from previous days and let the user know about this. The application can include intelligence to detect patterns in the user’s behavior and bring this to his attention. For example, if the user is frequently failing to complete about one hour of tasks per day, the application can suggest that the user commit to fewer hours per day. The application can also compare words in tasks to find similar tasks and learn patterns based on them. For example, if the user frequently fails to accomplish tasks with the words “Take out trash” in it, the application can let the user know.

An even more advanced system could set up a game layer to the application with unlockable accomplishments. For example, if the user completes all of their tasks on time in a given week, the application could acknowledge this and reward the user with a virtual trophy. Each user could then collect these trophies. If the application costs money, it could even reward the user with a small discount on his subscription for achieving certain milestones. The financial incentive is one way to encourage the user in addition to the competitive layer of collecting these trophies.
These virtual trophies could be conferred for the following accomplishments:

1. Completing all tasks on time in a given week, month, or year

2. Completing all tasks in a project

3. Setting a personal record for the most tasks in a day, month, or year

4. Setting a personal record for completing the greatest duration of tasks in a day, month, or year
6.1 Missteps with Complicated Frameworks

In late November, the project was based on one of the many available sophisticated frameworks for developing interactive web application, such as Cappuccino and Ext JS. These frameworks claim to make “it easy to build desktop-caliber applications that run in a web browser” (Cappuccino, 2012). Unfortunately, many of these technologies are very new and have limited documentation and community support.

At first, however, they seemed very attractive. Cappuccino, for example, was used to build 280 Slides, an impressive web application like Microsoft PowerPoint that runs in a web browser but looks and feels like a native application. Through the promotional information on the Cappuccino website, the developers of this framework appeared to know precisely what a developer’s pain points were going to be, so it seemed like a good idea to abandon the previous work, which would not easily commute to the new platform, and try developing using one of these frameworks. Unfortunately, this would prove a costly risk.
Figure 6.1: 280 Slides is a web application for developing presentations that uses the Cappuccino framework. It contains many features and an interface one would expect to find only on a native application, not in a web browser.
Figure 6.2: This mockup, designed in Photoshop, represents an early concept of what that application was to look like.

The framework chosen was Ext JS 4.0. The product seemed mature; the features, robust; and the plugin for compatibility with mobile devices, exciting. This particular framework began as an open source project with an LGPL-style license, but it merged with other software companies in 2010 and became part of Sencha. Sencha seems to have targeted the framework at commercial developers, as it has a sophisticated licensing system and a large price tag. While there was a free license for personal use, a single corporate license costs $595.00, and a license for 5 developers is a hefty $2695.00.

This high price implied that this framework was very powerful. The list of clients and testimonials also seemed impressive. If so many of these companies were willing to pay to use this software, it suggested that the framework was a good choice.
The Ext JS Chapter of Development

The EXT JS personal license includes access to the free online documentation, which seemed complicated but manageable. The application was intended to appear as shown in Figure 6.2, and EXT JS handled some rather sophisticated formatting problems with ease because the formatting is designated using JavaScript rather than CSS. CSS is a way to specify rules the browser tries to follow to present the page to the user, but with JavaScript these rules can be more complicated commands with logic. EXT JS provides a way to use nested layout schemes with variable and fixed dimensions to create a precisely controlled interface. When the window resized, EXT JS went to work and recalculated all of the proper dimensions for the components to keep the application in-line with the specified commands. EXT JS handled this all beautifully, as shown in Figure 6.3.

However, EXT JS's strengths were limited. Building functionality on top of this layout proved far more challenging. The example on which the formatting had been based appeared not to be compatible with the examples online for how to add functionality such as drag-and-drop. The application was configured as a Viewport, meaning the application took up the entirety of the page. Other applications are simply integrated within pages. The examples provided in the online documentation were not using this Viewport setup, and it was not clear how to make these examples work with the Viewport setup. Not only that, but the documentation was often outdated or limited. The articles were frequently for older versions of the framework, and no indication was given regarding how compatible these versions were. There were a few video tutorials available, which seemed well done. The interface made it seem like there were many more of these videos, but looking further made it apparent that these were just for show. In truth, there were only a handful of them, and beyond that, there was not much other depth.

It seems that Sencha did not care much for those using the free license. They
Figure 6.3: This basic structure was generated by Ext JS using a programatic approach.
offered numerous training courses and additional resources, but these were for addi-
tional cost. After a few weeks of refactoring the code to work with EXT JS, it became
apparent that this was going to be a dead end. Unfortunately, this meant two months
of lost work trying unsuccessfully to get EXT JS to work.

6.3 Back to jQuery

It became clear after this episode that the ideal way to develop the application would
be to use software that had good online documentation and a vibrant support com-
munity to back it up. For this, jQuery seemed the obvious choice.

jQuery is a JavaScript framework that is free, open source software. It makes
many complicated tasks relatively straightforward, and it integrates nicely with other
JavaScript libraries. jQuery UI is a related library, although separate from jQuery,
which includes functionality for user interfaces, including drag-and-drop, which is
critical to this project.

The framework has been an integral part of the web community for a few years.
In fact, it has gained rapid community adoption. Over 50% of the top 10,000 websites
online use jQuery, and its adoption has been growing at a similarly fast rate across
the top 100,000 and top 1 million websites, as shown in Figure 6.4.

Using jQuery required developing sophisticated structures and algorithms to han-
dle the complexity that the frameworks like Ext JS might have handled (such as
ensuring that the interface and the data stores are synchronized), but jQuery is far
more accessible, and this provided an opportunity to exert significant control over
the application’s underpinnings. Millions of developers have used jQuery to build
their web applications, and a vibrant community has developed around this library
answering common questions and providing thorough and up-to-date tutorials.
6.4 Technical Challenges

Once underway using jQuery, there were a number of major challenges to overcome to create the application. The way in which these were managed has significant implications to the capabilities and performance of the application.

6.4.1 Data Storage and Synchronization

One of the primary and most fundamental challenges of building a web application is keeping the data synchronized between the server and the client. The application will store data in three locations: in JavaScript objects, in local storage on the client, and on the server.

Various Locations of Application Data

The first location is data stored on the client in JavaScript objects. The application stores the state of the dates being shown within the window. This means that as the page loads for the first time, the application loads into local memory on the client...
(within the browser) the information regarding the days being shown on screen. This includes data regarding the tasks assigned to each day and anything else necessary to show the page, such as the list of projects and any settings saved from a previous session. The application structures this data using JavaScript objects so that the page can then be displayed.

When the user moves to different days, the application loads from the server in realtime using Ajax the state of the dates to be shown. After this, it updates the application’s interface. A possible enhancement to the responsiveness of the application involves caching the state of dates directly before and after those shown on screen so that the application can easily switch to the next or previous week and display the tasks without needing to request the data, a potentially slow operation.

As the user makes changes, these changes are reflected in the JavaScript application immediately, and then they must be synchronized with the server. However, before sending this to the server, the application stores the data in local storage, which is very fast compared to storing the data with the server and sending it over the Internet. By storing the changes locally at first, this can prevent disruption if the user’s Internet connection becomes suddenly unavailable or is otherwise unresponsive. Even if the Internet connection is lost, the changes will have been saved locally, and the user can continue working. The application can synchronize the user’s changes with the server when a connection is reestablished.

HTML5 introduced the concept of local storage within a client. In this new feature, web applications can store up to five megabytes of data on the client. This data is saved until the application or user deletes it. Unlike what can be stored using JavaScript objects, this data is not deleted if the page is refreshed or closed. The specification permits applications to store data in a simple key/value dictionary. A major limitation is that the data must be stored only as string values. JavaScript objects cannot be stored directly. Instead, they must be serialized.
An efficient way to do so is to serialize the objects using JSON, short for JavaScript Object Notation – converting the objects into a human-readable format which can later be parsed to reconstitute the objects. JSON is a format for interchanging data, and it is especially lightweight compared to XML, which requires tags. JSON is also human-readable, while XML is somewhat less so. This encoding format is based on a collection of name/value pairs, and it can be used to represent data in such a way that many languages can understand the data. In this application, JavaScript and Python exchange very complicated data structures over HTTP encoded as JSON.

Once the data is stored locally, the application attempts to store the data on the server for even more permanent storage. The application uses an Ajax request to do so, sending data serialized using JSON. The server then deserializes the data, checks that the changes are valid, stores them to the database, and sends back to the application a response indicating success. The application then updates the local storage to reflect that the changes have been saved (so the application will not attempt to synchronize them again) and display a message to the user. This transfer of data is shown in Figure 6.5.

**Data Transactions Between Server and Client**

There are a few main data transactions that need to be accounted for. The first is the initial load, when the client application requests data it needs to render the application and present the state of the application to the user as it was last saved (and potentially modified by the software in the interim since this last save). In this case, for TDLoo, the server must send to the client the list of tasks and all of their metadata (e.g. due dates, durations, etc.), the projects the user has created and their metadata (e.g. their colors), and any other settings (e.g. the number of days to show in the viewport). To facilitate this transfer, both the server and the client must know how properly to exchange this information. That is to say that the client must be
Figure 6.5: The browser and client are separated by the internet, and they communicate using Ajax via the jQuery library.

able to understand and act on what the server sends it, and the server must be able to communicate this information to the client upon request.

**Server-side Data Models**

The data in this application is stored on the server according to the following data models and fields:

1. UserProfile
   
   (a) First Name
   
   (b) Last Name
   
   (c) Username
   
   (d) Email
   
   (e) Password
2. TaskList

(a) Name of list

(b) Owner (the user that owns this list)

(c) Delegates (other users with access to this list)

(d) Date and time created

(e) Active flag (whether this is in use or archived)

(f) Last action record (pointer to most recent action)

3. ActionRecord

(a) List this action is affecting

(b) User who committed the action

(c) Date and time of action

(d) Unique hash (representing state)

(e) Direction (e.g. Do, Undo, or Redo)

4. ActionField

(a) Record (pointer to ActionRecord to which it belongs)

(b) Key

(c) Value

5. Task

(a) List to which it belongs

(b) Title of task

(c) Duration (in minutes)

(d) Do date (when it will be done and where it appears on calendar)
6. TaskColor

(a) Title
(b) Red value
(c) Green value
(d) Blue value

7. Project

(a) List to which it belongs (pointer)
(b) User who owns it
(c) Title of list
(d) Color of associated tasks
(e) Order in list of projects

6.4.2 Undo and Redo

An important feature of any application is the ability to undo and redo actions. This enhances the interface because it encourages users to learn without worrying that if they make a mistake it is not reversible. By assuring the user that anything they do can be undone, the developer encourages the user to feel comfortable using and exploring the application.

Implementing undo and redo on a web application, however, requires quite a bit of custom design, as there is no built-in library or API that makes this easy. This
is because undoing or redoing an action depends strongly on the application itself. For example, in a web application, it is not as simple as rolling back the DOM to a previous state to execute an undo. The application needs to update its local data stores, perhaps even those on the server. This means that undo and redo need to be custom-built from the ground up. That requires adding significant complexity around each action that can be undone. There does not appear to be any consensus on how best to do this, so the following approach was taken, which has proved effective.

Each action is packaged as an token, or object, containing sufficient information both to do and undo the action. This includes three essential components: what field is being changed, what the original value was before the change, and what the new value will be after the change is applied. In this way, the application can use these self-contained tokens to apply or undo actions.

Actions taken by the user in the application generally involve their interaction with some jQuery-enabled activity, such as resizing or moving a task. These actions have callback functions associated with them, code that executes when the application notices that the user has taken an action. These callbacks are set up to create action tokens. Within the callback function for a specific action, the application gathers information about what changed and creates an instance of an action token. Since the state of the application is rapidly changing, each state of the application is assigned a unique hash assigned by the server, and the hash of the state of the application prior to the action is appended to this action token so that the server can validate the change.

For example, after a resize event, the application determines which task changed, the original length of the task in minutes, and the length after the resize was completed. After a movement event, the application determines which task moved, from what day it originated, from which position in the task list it came in that day, to which day it moved, and to which position in the new day it was placed.
Each action has different parameters, both in number and order. This means action tokens are generic wrappers which must accommodate different types of tokens. That is to say, a resize action token will be very different from a move action token. Nonetheless, it is essential that these tokens are of a generally similar format so that they can be sent through a common pathway for processing. While there are many different sources of actions, there is a unique pathway through which all action tokens must flow. This is an invariant of the program that must be maintained in order for undo and redo functionality to work.

Once the action tokens are created, they can then readily be transacted with the server, as these tokens contain enough information to update the data models on the server and also to roll back (undo) the change that was just made if the user wishes to do so.

The common pathway through which these action tokens flow is described below and depicted in Figure 6.6 (assuming a network connection to the server is available).

1. Update the model of the data within the application
2. Append the action to the end of the list of recent actions in the browser’s persistent local storage
3. Mark the action as stored locally
4. Send the action to the server using Ajax, and receive a response
5. Determine whether the response indicates success or failure, and display a message to the user
6. If successful, mark the action as stored on the server

If an Internet connection is not available, the application will do the following:

1. Update the model of the data within the application
Figure 6.6: Action tokens are sent to the controller, stored locally, sent to the server, compared against the server’s records to determine validity, stored on the server, and then confirmed to the client.

2. Append the action to the end of the list of recent actions in the browser’s persistent local storage

3. Mark the action as “stored locally”

4. Wait until an Internet connection becomes available to synchronize with the server

6.4.3 Offline Mode

This application is designed to be capable of functioning with limited or no Internet connectivity. Even though Internet access is rapidly becoming ubiquitous (even some airplanes flying at 37,000 feet now have WiFi), designing an application that can function offline is not just an interesting exercise but also valuable because it adds significant robustness to the application. Developing an application capable of
functioning offline is not straightforward, as there are many decisions to be made without clear answers. An important question is how much functionality to maintain when the application goes offline. Another question is how to handle synchronization collisions (when the client running offline’s data conflicts with that of another client that is running online). In addition, should the application be able to initialize when the user has no Internet connection, or is it acceptable to require that the application was open before going offline?

To implement this offline functionality, the state of the application must be saved locally and all user changes need to be saved in a queue. Typically, these changes will immediately be sent to the server and saved. However, for the application to work offline, these changes need to be tokenized and potentially stored until they can be synchronized. The HTML5 local storage functionality is especially helpful, as it can be used to store these data, even across multiple sessions, synchronizing with the server when a connection to the Internet becomes available.

While the application is not currently enabled for offline access, the action-token-pathway architecture that was necessary for undo and redo functionality makes implementing an offline mode rather straightforward. To support an offline mode in which the user can make changes that will be stored to the server once the Internet connection resumes, there are two possible ways to do this.

The first strategy involves allowing the user to change the state of the application many times and then, once the Internet connection resumes, sending only the latest state to the server. This has the advantage of simplicity. However, there are two major drawbacks. First, without keeping track of what changed, the state of the entire application may be very large and costly to send over the Internet. Instead of just sending to the server what changed, the application must send the entire state of the application to the server. This could be overcome by implementing an algorithm that takes a “snapshot” of the state before going offline and compares this with the
state of the application once it comes back online and only sends the difference to
the server. While this overcomes the problem of sending large quantities of data back
and forth, it does not address the second problem, which is the inability to roll back
individual actions. In a state-based saving mechanism such as this, the application
does not keep track of what changed (although it could be computed if “snapshots”
of the application state were taken and maintained between changes). Keeping copies
of these states would end up being costly and redundant when all that needs to be
stored is what changed. Once again, this could be computed by comparing states,
but this is essentially the same as the second possible way of handling offline changes,
with action tokens.

Action tokens are self-contained units which describe exactly what changed for
each individual change. This presumes that changes can easily (or at least effectively)
be discretized. As the user makes a variety of changes to the application state, each
change is recorded individually and added to a first-in-first-out (FIFO) queue. The
state of the application on the client continues to update in realtime, and action tokens
continue to line up, waiting to be sent to the server when a connection is reestablished.
Finally, once the connection is reestablished, the action tokens are sent sequentially to
the server and applied one-by-one. This allows each action to be rolled back, action-
by-action, at a later time. The strength of this approach is that only the changes are
sent over the network, not the entire state of the application. This approach works
best when the application state is large and there are not expected to be many tokens.
For example, if the application is a word processor and each keystroke were a token,
this method would not work very well, as thousands of tokens would need to be sent
to the server upon network reconnection. However, in an application where, over the
expected timeframe for which the application will be offline, the user will make a few
dozen changes at most, this strategy works well. It maintains the ability to roll back
changes action-by-action and it involves minimal data exchange.
To make the current application work offline would require adding functionality that monitored for an Internet connection periodically and, when a connection was available, send the action tokens sequentially to the server. There are a number of small changes that would need to be made throughout the application to ensure that it works offline, especially handling the hashes that are otherwise created by the server after each action is stored to the server. These state hashes are vital to the application, and the application would need to be able to handle multiple actions (i.e. state changes) without getting an updated hash from the server.

6.4.4 Accommodating Multiple Simultaneous Browser Windows

Unlike native desktop applications which can limit the number of windows open, client applications can be open in multiple windows at once (even multiple windows on multiple computers), causing major problems for an application of this nature. If a user has two windows with the application open and makes changes in one window, the other window’s data has become out of date, or stale. For example, if a user deletes a task in the first window and then modifies that task in the second window, where it still exists, this presents serious problems of synchronization.

There seem to be two main ways to handle this: displaying errors when a user takes an action in a duplicate window, or synchronizing all windows.

To accomplish the former, one implementation involves the server and client exchanging random hashes indicating the last state of the application. Each time a change is made and sent to the server, the server generates a random hash which it stores and sends back to the client. The client sends the most recent hash it received from the server with every action it wishes to save to the server. Before the server applies the action to its data store, it checks whether the hash of the client matches the hash of the most recent state on the server. If these hashes match, it means that
the server and the client are in sync: no changes have been applied to the server by another client. If the hashes do not match, it means another client (i.e. another open window) changed the server in the meantime, and the new change cannot be applied.

This approach is attractive because of its simplicity, not just from the perspective of a developer but also from that of a client. Keeping multiple clients in synchronization with a server is very difficult because there is no way for the server to send updates to clients over HTTP, and the ways of simulating this are not very elegant nor efficient.

One way of doing so is called long-polling. The client sends an HTTP request to the server, and the server waits until it has something to send back before answering this request. After the server sends an update (by responding to the request), the client opens a new request and waits. These requests may time-out (i.e. the server takes too long to respond), meaning the request was wasted. The number of wasted requests can be very large, especially if the updates from the server are sparse. In this case, if no other window was open, all of these requests would be entirely worthless.

The most contemporary way of doing this is HTTP binding, or Bidirectional-streams Over Synchronous HTTP (BOSH). This involves major modifications to the server to support the Extensible Messaging and Presence Protocol (XMPP), which is what Google and Facebook use for their realtime chat functionality. The added complexity for this feature may not be warranted at this stage of development. It seems reasonable for now to expect that the user will have just one window open at a time. Perhaps for future releases of the software, more advanced algorithms can be used to keep many windows in sync.
Chapter 7

Business Model

*If history has taught us anything, it’s that Internet business models are like buses: If you miss one, all you have to do is wait a little while and another one will come along.*  
-Steve Krug

An important part of creating a successful application is developing a model from which to cover the expenses associated with providing the service to users. There are a number of ways to do so, which will be described and evaluated below. In addition to immediate profitability implications, these business models may also affect the overall value of the application, as the business models determines the predictability of the revenue stream.

7.1 Business Models

There are a number of business models, ranging from simply asking for donations to requiring payment upfront. When evaluating business models, it is necessary to consider them in the context of the application and its functionality as well as the software’s target market. Ultimately, the goal is to find a way of collecting revenue
that is likely to support the acquisition of paying users.

7.1.1 Advertising

One such business model is through advertisements on the web application, such as those within Google’s GMail. These applications are mostly unobtrusive, yet they generate sufficient revenue for Google to be able to provide the service to users entirely free of charge.

One major downside of advertising-supported applications are how they may detract from the overall user experience by adding unnecessary clutter or distractions. To be successful, advertisements have to get the user’s attention and distract them from what they were originally trying to accomplish. This can be problematic for web applications if users feel like they are being distracted from their original goal. Further, advertisements generally need to be conspicuous, so they are designed to stick out rather than blend in with the design. This can make the application unappealing.

This revenue source is also incredibly variable. There are many schemes for assessing the fees to be paid for advertisements, the main two being cost per click and cost per impression. In the former, the advertiser pays a set fee when a user clicks a link. In the latter, the advertiser pays a fee each time an advertisement is shown. In either case, the fees paid for advertising vary based on market fluctuations and on traffic to the site. This is difficult for the publisher to control and therefore subject to high volatility.

7.1.2 Donations

If the publisher is not interested in making significant profit, one way to support the application is to ask for donations. This means the publisher provides the entire application, and all of its features, free for use to anyone and includes a way for satisfied users to remit payment as a gratuity. The request for donations can look
like an advertisement (e.g. a banner on the website), it can be a nagging popup the user has to clear every time they close the window or navigate away from the page, or it can be a message displayed just on the homepage or login page. Because this revenue source is unreliable and unpredictable, it is better suited to individuals who are hoping to cover their expenses rather than companies looking to profit.

7.1.3 Trial Period

One way to provide potential users an ability to try the application and then buy it is to provide a free trial period. This trial period can be measured in time (e.g. one week of free use) or in actions (e.g. 20 tasks). Generally, this trial period should provide the user with all of the available features, and then, after the trial period is over, require that they pay to continue using the service. This model works well if the product is compelling, since if the user comes to like the product and rely on it during the trial period, he will be more likely to consider paying for it than if he had not tried the product in the first place. However, the application needs to convince the user to pay for it within this short timeframe. If the application is one that takes a long time to demonstrate its value and the trial period is not sufficiently long, an application using this method might miss opportunities to demonstrate its value to customers.

7.1.4 Freemium

A very popular model is to offer basic features for free but charge for users to use more advanced or robust features. This is called the “freemium” model, a portmanteau of “free” and “premium.” A major goal of the fermium model is to “get them hooked, so that they decided to pay to keep using the product or to take advantage of premium features” (Feinleib, 2012, p. 64).

For example, Dropbox, a popular file synchronization and sharing site offers 2GB
Figure 7.1: Basecamp by 37signals offers a 45-day trial period for its project management software.
of storage for free to all users. This is enough storage space for users to try the application and understand how it works. However, for users that have more files with which they would like to use Dropbox, they need to pay for additional storage space.

![Dropbox Pricing]

Figure 7.2: Dropbox offers multiple tiers of pricing, ranging from free to various levels of premium service.

This model works well because, like the trial period model, it allows the users to see how the application works and then decide whether to pay for it later. However, there are also special advantages to this model. For example, some users may never need to upgrade to the premium model. This allows the publisher to acquire many customers who actively use the product. With a free trial, the publisher loses all of those users that did not choose to pay for the software. However, with the freemium model, users that don’t pay can continue using the software. So long as there is a
sufficient number of paying users to offset the costs of supporting these free users, this can result in higher usage numbers for the application, which in turn can impress prospective users and drive more signups. If the application is ever considered as an acquisition target by a larger company, this may allow the company to demand a higher price upon acquisition because of this large established userbase.

The main drawback of the freemium model is that it requires having many users, since the conversion rate to paid customers is very small. With a typical conversion rate of 1% to 5%, this means the application needs to appeal to many users in order to generate sufficient revenue (Feinleib, 2012, p. 65). There is also always the risk that all of the users will enjoy the product offered for free and never upgrade. The model has a number of drawbacks, but this has not stopped it from becoming one of the most common models for startups.

7.2 Payment Arrangements

7.2.1 One-time Fee

One way to collect revenue is to require a one-time fee be paid by users, which entitles them to unlimited future use of the application. For native applications, or any applications that need to be downloaded, generally each new version of the software demands a new fee. However, with web applications, where updates to the application are automatically rolled out to all users, this would not be possible without artificially limiting users to the versions that they purchased.

That is not the only reason this method of collecting payment is not well-suited to web applications. Since web applications incur ongoing costs for the server and maintenance, this poses a problem where what the user paid upfront to use the application may at some point not cover the costs of serving that user. That is why this model is not very popular in modern web applications.
7.2.2 General Subscription

One way to overcome the ongoing costs of serving a web application is to pass these costs along to the consumer. Typically, this means charging the user a monthly or annual fee for using the software. Generally, these payments are set to automatically renew so that there is no interruption of service. The credit card on file is automatically charged the amount of a new period.

One advantage of this method is that it drives down the cost of the individual payments compared to a one-time fee. Rather than charging the user for an expected duration of use, with subscriptions the user pays for only as much time as he uses. This means it is cheaper for a user just trying the service, since only one month’s payment is required to try it. Because the monthly payments seem less intimidating, there is less of a barrier for users to sign up.

If the publisher is interested in being acquired, having recurring revenue streams through a subscription service is very attractive to investors. Whereas in the one-time fee model, the company’s ability to generate revenue depends on its ability to attract new users, with a subscription model, the company continues to monetize each user it acquires. “Venture capitalists like recurring revenue streams because they are predictable” (Camp, 2002, p. 87). This reliability of the revenue model can contribute to a higher valuation for companies being considered by investors.

7.2.3 Freemium Subscription with Service Levels

One way to improve on the general subscription model is to add service levels to the application, starting with a free version for certain users and offering more powerful features or greater access to users willing to pay. This method is beneficial because it captures a variety of market segments, ranging from those not willing to pay at all to those willing to pay substantial sums for the features they need.

This is the model employed by Dropbox, as shown in Figure 7.2. There is a plan
for users who can suffice on the free plan, which includes 2GB of storage. Other than
that limitation, the product is full-featured. The goal, then, is to let users install
the software, synchronize their files with it, and then get “hooked” and want to keep
using the software. After this point, they can upgrade according to their needs. They
can upgrade to the service level with 50GB of storage space or to 100GB. To be
successful, they need to have enough paying customers to support the customers who
are not paying for the service.

7.3 Implementing a Model in TDLoo

TDLoo would work well with a freemium model because it is an application with wide
appeal. There are millions of corporate professionals, academics, and home users who
could benefit from the application and might be interested in trying it. However, since
this application’s competitive advantage is its interface, this is the type of “feature”
that is best experienced firsthand rather than advertised. For this reason, it would
be important to let users actually try the product. That is why either a trial period
or a freemium model would work. However, since this application is also directed
at corporate professionals who would be willing to pay for the software, a freemium
model could work well if there are enough corporate users who would be interested
in purchasing the upgraded features.

Since corporate professionals are the conversion target for the freemium model,
the features behind the paywall of premium service should be those that would be
important to these users. For example, the premium service could include features
such as reminder emails, offline mode, and the ability to share tasks with colleagues.
In addition, mobile access could be limited only to those subscribing to the premium
service. Further, encrypting the transactions for security could be a special option
for a higher tier of service.
7.3.1 Running the Numbers

There are 55 million white collar workers in the United States according to State-
HealthFacts (2012). If 1% of these workers end up trying the product, and 3% of these
individuals end up purchasing the product, that amounts to 16,500 paying customers
and 550,000 users not paying.

It is difficult to predict how much it would cost to host the application. However,
to obtain a rough estimate, consider some example sites.

Reddit.com, is a popular site that ranks 124th globally in web traffic according to
Alexa.com. The site in December 2011 hosted 35 million unique visitors who loaded
2 billion pages (Martin, 2012). The servers that run this site, using Amazon Web
Services, cost $22,000 per month (Jedberg, 2012). This is a cost of $0.00063 per user
per month.

In addition, consider the web application DropSend, a web application for sending
large files over the Internet. When the site had 13,000 users in 2006, it was spending
about $1500 per month (Carson, 2006). Since this application needs to store and
transfer large files, this site would be expected to be paying a high cost per user,
which it does compared to Reddit. DropSend at this time was spending $0.12 per
user per month.

Given the huge difference in these costs per user, it becomes clear how difficult it
is to estimate the costs of a web application. However, considering that TDLOO will
be transferring only text-based data (and small amounts), its costs per user would be
expected to be close to those of Reddit (or at least closer to those figures than that of
DropSend). Assuming it costs $0.001 per user per month and there are 550,000 users,
the monthly cost of the site’s servers would be $550. This means that the annual
revenue requirements to cover the server would be $550, which, for an estimated
16,500 paying users, would make the application profitable at many different price
points. The break-even price would be $0.03 per paying user to support all of the
non-paying users. Therefore, if the annual cost of the application for these paying users is just $10, or $0.83 per month, the application would generate an annual profit of $158,400 (excluding non-server costs). Based on these assumptions, the break-even ratio is 833 non-paying users to 1 paying user. That means the application only needs to convert 0.12% of users to paying customers in order to cover the server costs. Given that Feinleib (2012) assumes that a conversion rate of between 1% and 5% is normal, this application would be expected to be successful since the required conversion rate is an order of magnitude smaller than what would be expected.
Chapter 8

Conclusion

*It’s not enough to be busy, so are the ants. The question is, what are we busy about?*

- Henry David Thoreau

This application has the potential to give individuals more control over their daily lives. It provides the functionality of a list so users will not forget tasks, a planner so they can set realistic goals, and a personal coach by giving them feedback on their accomplishments. Based on psychological principles describing how humans interact with software, the application is designed to require minimal cognitive effort to understand and to use. It combines the positive features of other similar web applications while avoiding some of their shortcomings. Various market segments, ranging from corporate professionals to academics to home users can benefit from the functionality the application provides. Based on a freemium pricing model with a $10 annual fee for unlocking more features, the application could be profitable even with a very low conversion rate from free users to paying customers. Therefore, this project should be pursued further.

While the proof of concept demonstrates the application’s potential, it still needs further development before it can be launched. For example, the prototype demon-
strates how the interface works, but the reminder emails and motivation tools are not yet in place. Also, the application is not yet ready to support multiple users. For the purpose of testing, it is limited to serving only one user. The architecture is such that it can serve multiple unique users, but there is no interface yet for creating accounts and user authentication. Even without this infrastructure, testing can begin on the interface.

Observing how testers interact with the application can bring to light design problems that would not be apparent otherwise. For example, if multiple testers cannot figure out how to do something, such as move a task, this would suggest that should this be reconsidered. While attempts were made to follow the principles of good design according to the literature on psychology and interface design, it may be the case that the implementation of these principles could be improved. Such testing can also reveal defects in the program that were not previously identified.

After this interface testing is complete, a second phase of testing should consider how the application impacts users. This testing should include participants first trying to organize themselves for one week with another tool, such as another web application or a conventional paper organizer. Then, it should involve the participant trying this web application for a week. The difference between how they use the tools and how they ultimately feel after using each will provide information about how the application compares to those it was designed to supersede. For example, the participants may find this application’s interface easier to use, which might result in them using the application to record tasks they otherwise might not have with a different tool. Therefore, it would be expected that the participants would record more tasks using this application. Similarly, a comparison can be made considering the quantity of tasks that the user ultimately completed. If it were found that participants ended up completing more tasks using this application, this would support the hypothesis that the application helps individuals accomplish more. Finally, the participants’
opinions on how productive they felt using each tool would be useful in considering how this application compares to its predecessors.

While this can be an iterative process of design and test, it would be ideal to get an even wider range of feedback by moving to a public beta. By launching the application as a beta and allowing anyone on the Internet to test it and provide feedback, more perspectives are added and the quantity of feedback increases. In addition, doing so could garner media attention that would be helpful in promoting user adoption.

At a point where the application is sufficiently stable, the beta moniker can be removed, and the application can launch with a business model in place. Users paying for the service would be able to access special features such as encryption and the ability to set up daily reminder emails. Even if only 500 users pay for the service at the proposed $10.00 per year freemium model, this amounts to $5000 of annual revenue, which would be enough to cover the costs of the application.

Therefore, it would be worthwhile to take the necessary steps to bring TDLoo to market. Individuals like Brad, Jeff, and Molly would benefit from how this application helps them organize themselves. Perhaps a major software company would take interest in the application and purchase it, exposing an even larger audience to the benefits of this application. This new application provides an innovative way for individuals to manage their time, to be more productive, and to have more time left to do what they choose to do rather than what they have to do. As Thomas Edison recognized, “Time is really the only capital that any human being has, and the only thing he can’t afford to lose.”
Bibliography


