Solutions to mitigate input drawbacks of Expense Tracking Application

Pooja Asher
North Carolina State University
pmasher@ncsu.edu

Zeal Ganatra
North Carolina State University
zganatr@ncsu.edu

Pratik Mukherjee
North Carolina State University
pmukher@ncsu.edu

Ashwin Oke
North Carolina State University
aboke@ncsu.edu

Sandeep Samdaria
North Carolina State University
ssamdar@ncsu.edu

ABSTRACT

Software applications are being used for almost all the purposes. Similarly, people use software applications to track their expenses. However, user survey shows that current input methods decrease accuracy and increase input time. The paper suggests four ways to improve the input via better User interface, Image OCR, Voice API, or via SMS, and also developed the prototypes for the same. It also reflects the telemetry that have been used to collect the data to evaluate the solutions. From the data, various metrics are built and then the solutions are evaluated using the metrics and comes to a conclusion that input via SMS is the best input method.

1. INTRODUCTION

Technology is omnipresent in daily aspects of our life. Everything is being automated. On similar line, expenses sharing has been automated. Applications are being developed for the same. Some common applications used for expense sharing and tracking are Splitwise, BillPin, Reciept Ninja, etc.

From the study performed on vast number of users revealed that manual input in expense sharing application increases the probability of input error and input time. The paper aims to propose a solution that reduces the input error rate, increase accuracy and reduce the input time. For this purpose, we developed, tested and compared four prototypes. Each prototype provided a different input method using better User Interface, Image OCR, Voice and SMS respectively.

The paper initially gives the methodology followed in the development of the project. After that the working and technical requirements of the four prototypes are explained. We then explain the criteria and method of comparing the input methods. Finally, we conclude that every input method has its own pros and cons. However, the input via SMS and via Voice, are more user favored. But the SMS input gives better accuracy and least input time. Therefore, considering all the constrains we finally suggest SMS as the best method for input.

2. METHODOLOGY

2.1 Project Software Development Model

We have followed an agile software development methodology setup in the project. The primary reason for doing so was to have some flexibility in our development process. We did not want to have rigidity in the different phases of software development as we were not sure of the exact business and technical requirements upright. Agile allowed us to have several iterations of design-development-testing cycle. We extensively used GitHub for project code collaboration and management. We used GitHub features such as GitHub labels, milestones and issues for project management. Each task was tracked as a GitHub issue. We had different labels generated in GitHub attributed to different individual items involved in the project. We tagged an issue to different labels, so as to know what an issue is concerned about at a high level. We had multiple milestones in GitHub to track the overall project progress. We followed development practices such that every code check-in gets tagged to the issue it addressed.

2.2 Architecture

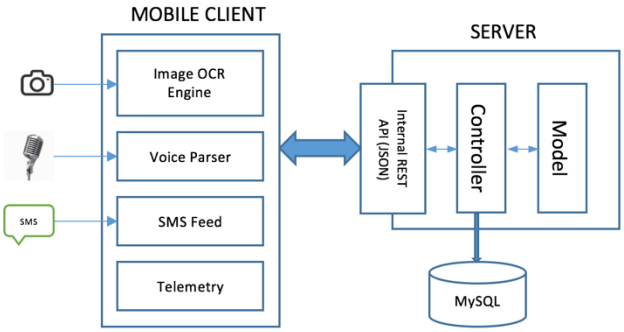

Fig 1. Architecture

We identified that our project had different individual components. Following are the different components:

- Four solutions: The four different solutions could be regarded as independent pieces of this product. Each solution can further be treated as a standalone component for development purposes. Below are the four solutions:
  - Voice Parser
  - Image OCR
  - Improved Manual Input
  - SMS feed
- Base Android setup: This component dealt with development of android workflow, including taking inputs from the user and saving the expense with those details. Upon a "save" event, a REST call (POST request) is made by the application to the back end. This also includes an authentication module, responsible for login and logout functionalities.
- Back end framework: This is the back-end component which exposes a RESTful API for adding an expense. A MySQL database resides in the back-end to store the expense data. We used Django, which is a high level Python Web framework to power the back-end to expose the REST API.

2.3 Tasks Division

As noted above, we identified the several different independent components that we had in our project. We divided these independent components amongst ourselves such that each of us would be the POC (point of contact) for the respective thing he/she is working upon. We decided to build a sample working prototype for each of the solutions (in the form of small Android apps). Once the build was over, we unit tested these modules thoroughly and once passed, the idea was to integrate these individual pieces to the main workflow. System Integration was a collaborative process where each of us contributed to add the individual component into the final product. Once this was done, we together performed system testing on it, before it could be pushed for user testing.

3. IMPLEMENTATION

3.1 Improving UI for Manual Input

The analysis of the results of the user observation reveals a major drawback of most expense sharing mobile applications: they don't have an inbuilt input keyboard through which the user can provide an input about the expense and the description. Most of these applications rely on the soft QWERTY keyboards of the mobile devices and the platforms on which they were installed for providing the input. Predictably, the user interface of these soft keyboards varied considerably across the platforms and the mobile devices. The lack of a standardized user interface for the input mechanism meant that the accuracy of the user input and the time taken was subject to the quality of the design of the manufacturer. A major deterrent of these soft keyboards are that they cannot be customized for the different applications based upon the need. Alphabets, digits, special symbols and punctuation symbols are often congregated together in a single layout often occupying less than bottom half of the screen. This reduces the key size of an individual component which often makes the usage to be more error prone.

Only a small subset of the components present in a soft QWERTY keyboard are used in an expense sharing application. Expense is numeric by nature and its precision is mostly restricted to two decimal places. The entry of the expenses requires numeric digits (0-9) and a decimal point only. The entry of the description requires just the components for alphabets and a small subset of the punctuation symbols. None of the special characters or symbols in the QWERTY keyboard were required by an expense sharing application. This meant that if we can design an inbuilt input interface for a user with just the required components, it could result in a much enhanced, user friendly design that could reduce the errors of manual input and mitigate the disadvantages of a soft keyboard.

Henceforth we decided that an inbuilt minimalized user interface customized for expense sharing usage can solve the problems of the incorrect user input of the expense sharing application. We also decided to use separate sub-interfaces for the expense amount and expense description as both of these require standalone and exclusive input components as previously mentioned. Two different interfaces meant that the input components would be separated that would result in an increase in the component size and a better design.

As a bulk of the users had Android devices we decided to implement our solution for Android platforms. As per the implementation the user would be prompted to enter the amount of the expenses that he/she intends to share through a customized user interface that had the numeric digits (0-9) and the decimal point. The user also had the option for deleting an incorrectly entered mount, resetting an incorrect amount altogether. The interface also restricted the entry of an invalid numeric input (Example: Numeric amount with two decimal points) by preventing the navigation unless a valid amount is entered. On entering a valid amount, the user can navigate to the next user interface for entering the description of the expenses to be shared. This would prompt the user to enter the description through a soft keyboard that has the alphabet components. Upon entering the description, the user can save the details of the expenses in the application. The solution wasn't integrated with an application as the mechanics of it would be subjective to the nature of the application and would considerably vary.

Even though the customized interface with a better design and separation of concerns resulted in an ease of user input, the solution had some obvious drawbacks. For one, it still relied on the manual input to be made by the user and didn't introduce any automation component. The solution would also not provide any significant gains for people who are not comfortable with manual input. Also two separate interfaces meant an increase in the navigation time from one to another and delay in the entry of the expenses involved. Therefore, alternate solutions that could leverage the automation of the user input and reduce the delay in the user input were thought upon.

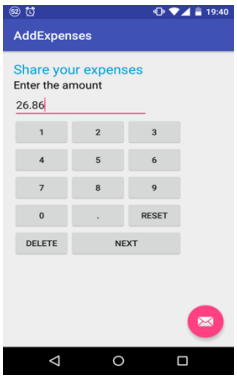

Fig 2. Improved Manual Input

3.2 Input via Image OCR

One of the alternative of manual Input is using Image as a input. From every purchase the user has bills, from which s/he inputs the amount using the manual input, but it is prone to errors. The image of the bill can than be used as a input, this reduces the time needed by the user to read the bill and input the amount.

We have implemented the this using the Tesseract OCR [1]. Tesseract is an open-source software, released under the Apache License, Version 2.0, and development has been sponsored by Google since 2006. Tesseract is considered one of the most accurate open source OCR engines currently available.

The application captures the image, preprocesses it and feeds it to the OCR Engine. The preprocessing of image involves increasing the contrast between the text and background. Preprocessing is necessary as less contrast gives a blurry image which increases the chances of faulting converted text from the ocr. Studies have showed that the percentage of accuracy for preprocessed image is hundred times better than the un-preprocessed image.

The OCR converts the image to text. The converted text is auto filled in the text field. The OCR needs to be trained from the text data give along with the OCR. The training data needs to be saved in the device along with the application. Due to limited training data the OCR is not very efficient. The training data has many variations of alphabets and digits but it has limited special characters. Using

this training data the OCR performs a supervised learning. The input image is then classified using this trained model.

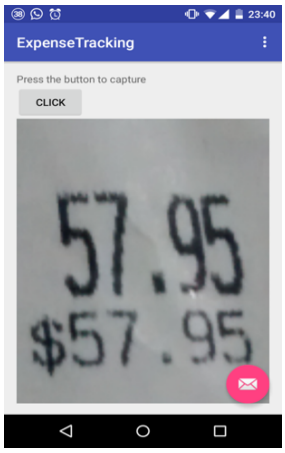

Fig 3. Capture Image to OCR to parse amount

3.3 Input via Voice

One of the most tested input forms recently is Voice. For the expense tracking application, the accuracy of numerical inputs is necessary. Using voice inputs is easy for users who can simply read bills or expenses and update them on the app. It is also a very natural and logical way of interaction for the average user, and aligns with Nielsen's heuristic of 'Match between system and the real world' [3]. The input for this solution can populate the numerical amount of the expense as well as the description of the same. Using statements like '120 dollars at the grocery store' or '67 dollars' or simply '35', the application can capture all given knowledge and also provide an option for manual input for the rest of the information. The voice expenses option opens in a new window where voice input is automatically taken. If the user finds an error or wants to change the input, he can also retry by using the small voice input button on the page. Once the user is satisfied with the input, they can move forward and will be redirected to the systems normal manual input system. If there is any inaccuracy in the voice input, it can be corrected using the manual keypad.

We have implemented this solution using the Google Recognizer Intent [4] for Speech. It has adequate accuracy and works well for Numerical inputs using voice as well. We run a few checks on the input recorded to make sure that numerical data is recorded. Also, the amount is separated from the description and passed as different fields in an Intent. The voice solution is hence easy to use as well as fast and accurate.

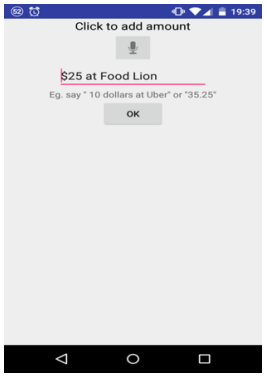

Fig 4. Voice Input

3.4 Input via SMS

An alternative solution to track expenses could be to extract expenses from the message feed of the device to look for any expense incurred messages. Since this application is an Android device deployment, SMS inbox can act as a very important source for expenses. In today's era, various banks and credit card agencies offer real time notifications to their customers in the form of SMSes in case of any debit/credit event. Expenses can be seen as debit events and we can extract expense details from the SMS notifications received from banks and credit card agencies. The crux of the solution is to scan the user's SMS Inbox for expense matching messages, extract the message details for expense amount and expense description, and finally feed these details to the application. Android provides Java APIs to retrieve the contents of an SMS Inbox. Next, we iterate over all the messages to look for expense related messages. We used a regex pattern to match to messages containing dollar amounts in them. Such messages would be recognized as potential expense messages by the application. The user is provided with a list of such matched messages in the application. When the user selects a particular message for an expense entry, the amount and description fields in the application are auto-populated with the details. Once the expense is submitted, the details are stored in the backend using a REST call exposed by the application back-end software.

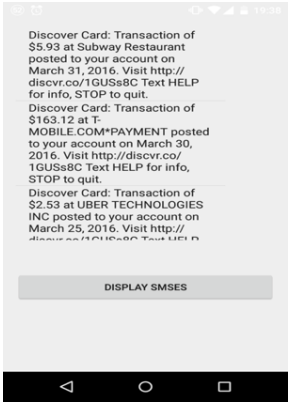

Fig 5. SMS List

4. EVALUATION

4.1 Telemetry

Since the users can use any of the above solutions we would require a framework which would collect metrics for each of the above solutions and report it to the backend server. We built a telemetry framework which would collect the metrics from the application automatically and unobtrusively and report it to the server. The telemetry captures both the high-level information as well as the granular level information from the application. The high level information that are captured are the solution usage and the time taken for the application. Granular information such as the number of keystrokes made by the user are also being recorded. Combining both these kind of information gives a deep insight into the application usage and helps us in evaluating each of the solutions.

Each expense record is associated with the type of solution used. For e.g. if a user uses voice to record an expense, then we tag the expense type as "voice". This metric "type" can help us identify the most common solution being used by the users. Another metric that we track is the total time taken by the user to store an expense. Once the user starts the workflow to add an expense. The timer runner keeps on running till the user saves the expense and the total time

is recorded. The "duration" metric in the expense records provides insights on how much the users spends time for each solution. The above two telemetries can give a high level information of the individual solutions.

To get a deep insight on how efficient the solution is, we track the number of keys users had to press for each solutions. While entering the amount and description via manual input, the number of keystrokes will be at least the length of the input. It is highly possible that the user can make mistake which can increase the number of total keystrokes. Since the application auto-populates the amount and description text via Image OCR, SMS, and voice, it is expected that user will avoid the hassle of manually keying in the amount and description and ideally will have zero keystrokes. It is possible that the value auto-populated by the solutions might not be accurate or would miss certain information. To track such cases, we track the number of extra keystrokes to fill in those information. The keystrokes tracked are categorized into two categories: a) Edit key-stroke count: The number of new key-strokes to add a new character/digit. b) Delete key-stroke count: The number of delete key-strokes to delete a character/digit. This metric can be used to identify the effort that the user has to put in terms of extra key-strokes.

4.2 Testing

Unit Testing: Every module of the solutions viz. the manual interface, SMS, Image and Voice based solutions were thoroughly tested as they were developed. The modules were incrementally built and each feature of the module were tested. Issues were logged in github to keep track of bugs and were closed as the bugs got fixed.

System Testing: After developing the individual modules, they were integrated into a single application and the functionalities of the integrated application were collectively tested. Issues were logged in and fixed as previous.

User Testing: 20 users were selected for testing the integrated application. These users were selected from amongst the ones who had participated in the initial survey. The application was deployed to these users and each user was initially asked to test each of the four solutions for a minimum about 2 times. After the initial testing, the user was also asked to test any two of the four solutions that he/she was most comfortable in using. As a part of the user testing, the users were also asked about what they liked and disliked about each of the solutions. They were also asked about any features that could improve any of the four solutions. Based upon the results of the user testing and an assessment of the scope, the results were further improved to induct the desirable features as proposed by the users and to lessen and do away with the undesirable features of the solution. The results of thus testing were collected as a part of the telemetry process and they were analyzed to test the viability of the solutions.

5. RESULTS

5.1 Application Usage

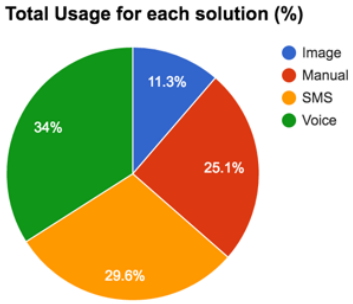

Fig. 6. Pie-chart representing the percentage usage of solutions

The pie chart indicates the percentage of usage of each solution by the users. It shows that the voice based solution was the most frequently used one, followed closely by the SMS based solution, manual input. The image based solution was the least frequently used one. As the frequency of the usage can be co-related to the ease of use it can be concluded that the voice based and the SMS based solution have the maximum ease of use and the image based solution has the least ease of use.

5.2 Average time for each solution

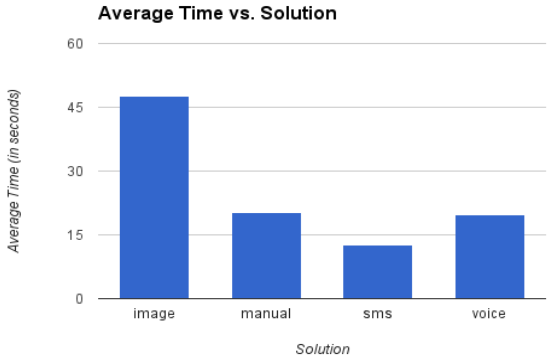

Fig. 7. Average Time for each of the solution

The bar graph depicts the average usage time for each solution. As per the graph the average usage time is least for the SMS based solution – about 18 seconds, followed by manual input – about 21 seconds, voice based input – 24 seconds and is maximum for the image based solution – about 34 seconds. This implies that the image based solution is the most time consuming input mechanism and the SMS based input is the fastest one.

5.3 Average time for each users

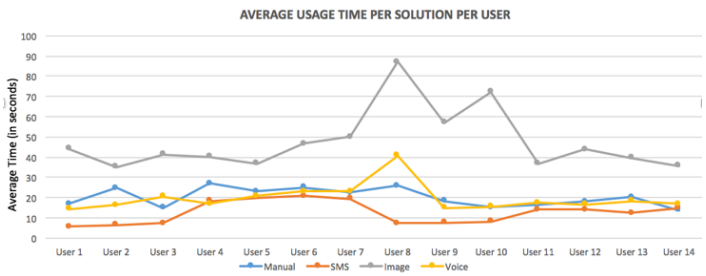

Fig. 8. Line graph representing average time per solution per user.

This is a line graph that depicts the average usage time for each solution taken by each user. It gives us an estimate regarding the variation of the average time taken for each user to make an entry in the application. The graph shows the SMS based solution and the Image based solution shows a relatively low variation of the usage time. The voice based solution has the maximum range of variation indicating that the usage time is highly relative to the user who is using the application.

5.4 Keystroke efficiency

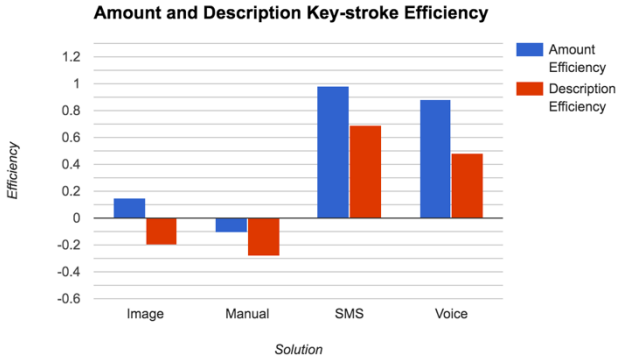

Fig. 9. Bar graph to indicate the efficiency for amount and description

The bar graph depicts the Key-Stroke Efficiency for both the amount entry and the description entry. The Key-Stroke Efficiency is defined as follows:

$$Efficiency = \frac{(Length\ of\ the\ input - Total\ keystrokes)}{Length\ of\ the\ input}$$

It indicates the number of manual entries or keystrokes "saved" for making an entry in the application. A higher keystroke efficiency indicates that that a solution is more accurate in terms of making an input to the application.

The graph shows that the SMS-based solution has the maximum keystroke efficiency for both the amount and the description entries followed closely by the Voice-based solution. The Image based solution has negative keystroke efficiencies for both amount and image solutions indicating that the number of inputs required were more than the actual length of the input that was made

5.5 User intervention factor

The graph depicts the Average User Intervention Factor for each of the four solutions for an amount entry. The user intervention factor is defined as follows:

$$UIF = \frac{\text{Number of Deletes or Edit}}{Length\ of\ the\ input}$$

The Average UIF for all the amount inputs of all the users was computed for every solution. The Average UIF was computed for both deletes and edits (the number of soft keyboard inserts). A less UIF indicates that the solution requires less manual intervention from the user for a correct input and therefore implies a better solution.

5.5.1 Amount

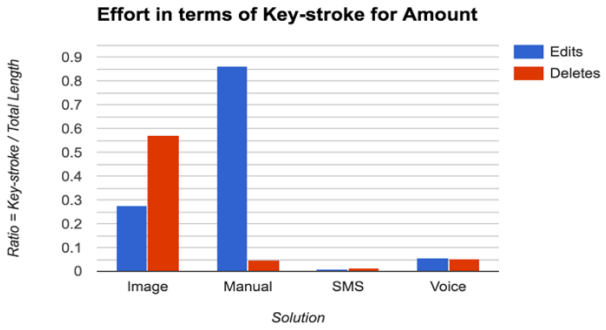

Fig. 10. Bar graph to represent the effort in terms of key-stroke for Amount.

The graph shows that the SMS-based solution has the least UIF for both edits and deletes followed by the Voice based solution. Manual entry through the designed interface has a low UIF- delete but a high UIF-edit ratio. The UIF is worse for the Image based solution for both edits and deletes. This depicts that the SMS based solution requires the least manual intervention for corrected input and the Image based solution requires maximum intervention from the user.

5.5.2 Description

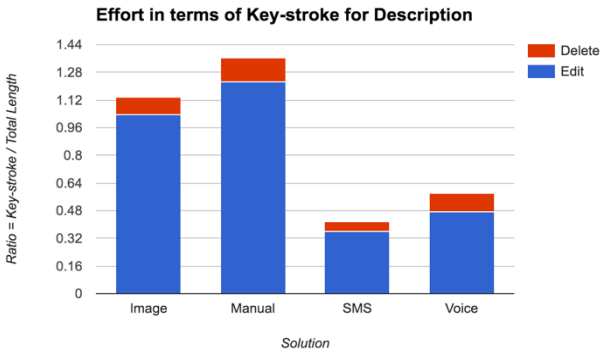

Fig Bar graph to represent the effort in terms of key-stroke for description.

The graph depicts the Average User Intervention Factor for each of the four solutions for description entries. The results also bear similar resemblance to that of Graph 5. The SMS based solution has the least UIF for both deletes and the edits – indicating that it requires the least user intervention for description inputs and the Image based solution has the maximum one which indicates that it requires the maximum user intervention when making an entry for the description.

Metric	Best Solution	Value
Most Used	Voice	34%
Minimal Usage Time	SMS	12.7 seconds
Keystroke efficiency - Amount	SMS	0.98
Keystroke efficiency - Description	SMS	0.67
Least UIF - Amount	SMS	0.01
Least UIF – Description	SMS	0.155

Table 1. Best solutions for the Metrics

6. CONCULSION

Our study revealed pros and cons of using the different input methods for numerical data. Using these methods on an expenses tracking application greatly improves its efficiency. It also directly relates to the time taken by the total time taken by the user on the application and in turn its user experience. This project was aimed specifically at input in an expense tracking application but it can also be applied to other domains which need numerical input data.

Our solutions showed that input via SMS and via voice showed the best results. In case of SMS, the time taken, overall accuracy and number of keystrokes required were all lowest. The issue with this solution is that it is dependent on a user activating SMS alerts for their expenses, and hence the usage of this solution was slightly lower in the overall study. In comparison, voice input was the most commonly used method for input, but slightly lacked in accuracy and took higher overall average time for inputs.

Input using image OCR had a few advantages including a feature to save bills permanently, and overcoming linguistic barrier. The issue with this method is that its accuracy is moderate, and needs more user intervention in terms of number of keystrokes required, Also, it took the highest amount of time overall. Better OCR systems are available as Software-as-a-service (SaaS) and these can be made available on the application for a fixed amount of money per usage. This is not ideal though since the user will have to be charged for the service which is a disadvantage.

Manual inputs are marginally better than the regular mobile keyboard used widely. Its accuracy and time is highly subjective on the users' typing skills and may also cause user errors while reading and copying amounts or while entering data. In spite of that, manual input is always used as a backbone for the inputs and provided optionally with each input.

Hence we conclude that SMS and voice inputs have the most merits as input methods for our expenses sharing application.

7. FUTURE SCOPE

Currently, the application is limited to add a new expense. We can build features to extend the application to view, edit, delete an expense. Currently, single item is tracked by the expense application. Itemized billing can be done to show the breakdown of all the items under a single expense. This can be achieved by using paid Image OCR tools which have higher accuracy. The application can have pro version which would display visualizations of all the various expenses and detailing the breakdown of all the categories.

8. REFERENCES

[1] Tessearact OCR for Android https://github.com/tesseract-ocr/tesseract/tree/master/android

[2] Python web-framework Django https://www.djangoproject.com/

[3] Ten Neilsen's Heuristics *https://www.nngroup.com/articles/ten-usability-heuristics/*

[4] Voice Recognizer Intent *http://developer.android.com/reference/android/speech/RecognizerIntent.html*