# Introduction

UXprobe is a company with one important question. How do you users interact with applications? From answering this question, the company builds the foundation of its business model, helping companies improve their applications. UXprobe tracks how users move throughout and interact with an application as well as gathering user satisfaction data. Through this movement and satisfaction UXprobe hopes to understand how to better applications.

# Dataset

The origins of the dataset come from one customer of UXprobe. This customer is a transport company. Every truck has a tablet that runs a software developed in-house. It runs on an Android platform. This software is mainly a navigations system (TOM TOM), but also manages communication between drivers and the company, the assignments of their tasks and time management (cf importance of the regulatory context, minimal resting time, etc)

Originally, the software was complex as final users (the drivers) were not involved in the development process. UXprobe system allows the project manager to analyze the use and satisfaction of the drivers with the system in context and lead to a redesign of the software.

## Principles

When a user starts interacting with the software, he enters a *session*. A session can last for several hours, typically 7 or 8 hours (due to the limitations of the time a driver can drive a truck before resting). The dataset contains a field ‘sessionID’ that uniquely identifies a session. During that session, the user will want to achieve several actions, such as sending a message. This type of action is called a *task*. A task can be successful: the task is completed, or unsuccessful: the task is abandoned or an error occurred. The field ‘taskID’ contains a description of the task.

To perform a task, the user will interact with the system, and perform some operations, these are called *events* (the field is actually called ‘activityID’ in the dataset, and contains a short description of the event in plain text). Events have different types: ‘screen’, ‘feature’ (the feature is actually called ‘event’ in the dataset), ‘error’, or other. Type is a field in the dataset.

When an event is started, a timestamp is recorded, and is stored in the dataset under ‘starttime’. The granularity of the timestamps has changed in the course of the development of UXprobe logging system. In the beginning the granularity was coarse, so that events could be timed at the same moment, which can lead to some abnormal sequence of events when sorted. A field sequence contains a unique identifier for the record in the dataset, and can be discarded for the analysis. In general, a task is a form of container for several events, but some events can happen outside of the context of a task (example: an error can occur).

## Data quality

In some events, the client inserted “(driving)” in the description, if the event happened while driving the truck. This string should be removed, otherwise it would create two apparently different events, while only one is meant.

## Missing information

The hierarchy of the features of the system is not available, the logs contain events, screens and features. The structure of the software (such as the hierarchy of menus, screens, etc) is not documented. In general, UXprobe does not necessarily know what the actual feature is, but the customer knows.

The actual sequence of events is not directly available in the dataset, but can be inferred from the timestamp: an event is followed by another event if it has a timestamp that immediately follows.

## Missing data

We have no information about the comprehensiveness of the data, and if they represent the most common use of the system, and if all records of the sessions received are included. From the userID field about 50% of the entries are missing. Around four hundred and fifty thousand events do not have a task.

## Volume of data

Two files of approximately the same size were available, with a total of 1.2 million records and 160,9 MBytes.

## Description of the entities and their corresponding fields

### Session

The field name is sessionID and contains a unique identifier of session

Example of value: ‘55df6ce6-bac0-4bbb-b7ab-2e824d8a01a4’

### Record

The field sequenceID contains a unique identifier of the record, and can be ignored.

Example of value: ‘3343ae5e-997d-11e4-bfb2-f7c03b184ea7’

### Task

A field taskID does not contain an ID, but a short description of the task.

Example of value: ‘New Message Notification’

### Event

The field is called activityID but contains a short description of the event rather than an ID. In this dataset we identified XXXX different events values.

Example of value: ‘Notification: new message’

### Event type

The field ‘type’ contains the type of the event, which can take XXX values: ‘screen’, ‘event’ (where here it is ‘feature’ that is meant), ‘error’, ‘task’ (??? What is this one?)

### User

A field userID contains the identification of the user.

Example of value: ‘VEACESLAV OK-000000000046Y000-327700050402453’

# Statistics of the Data

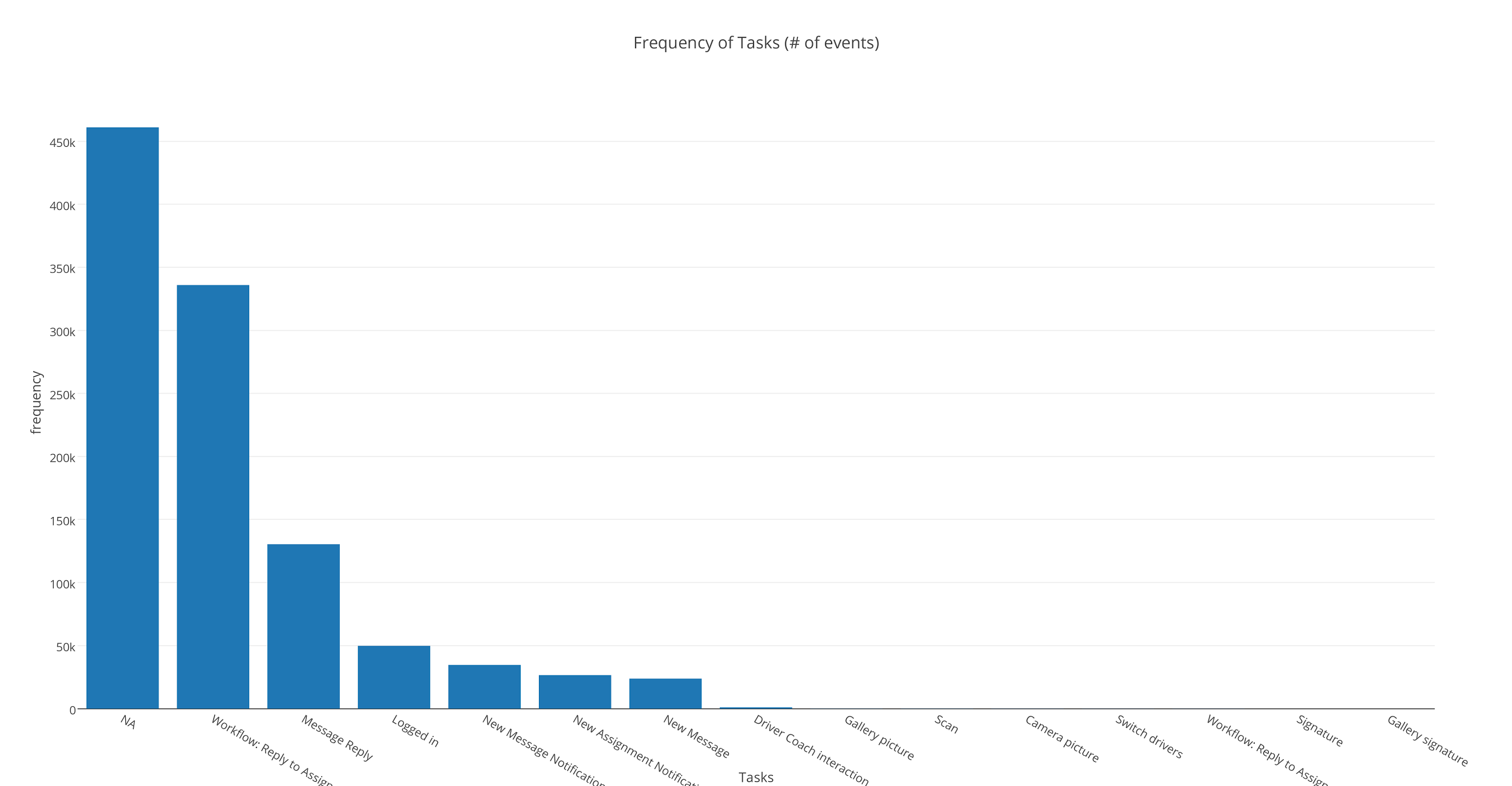


Figure 1 Frequency of Tasks

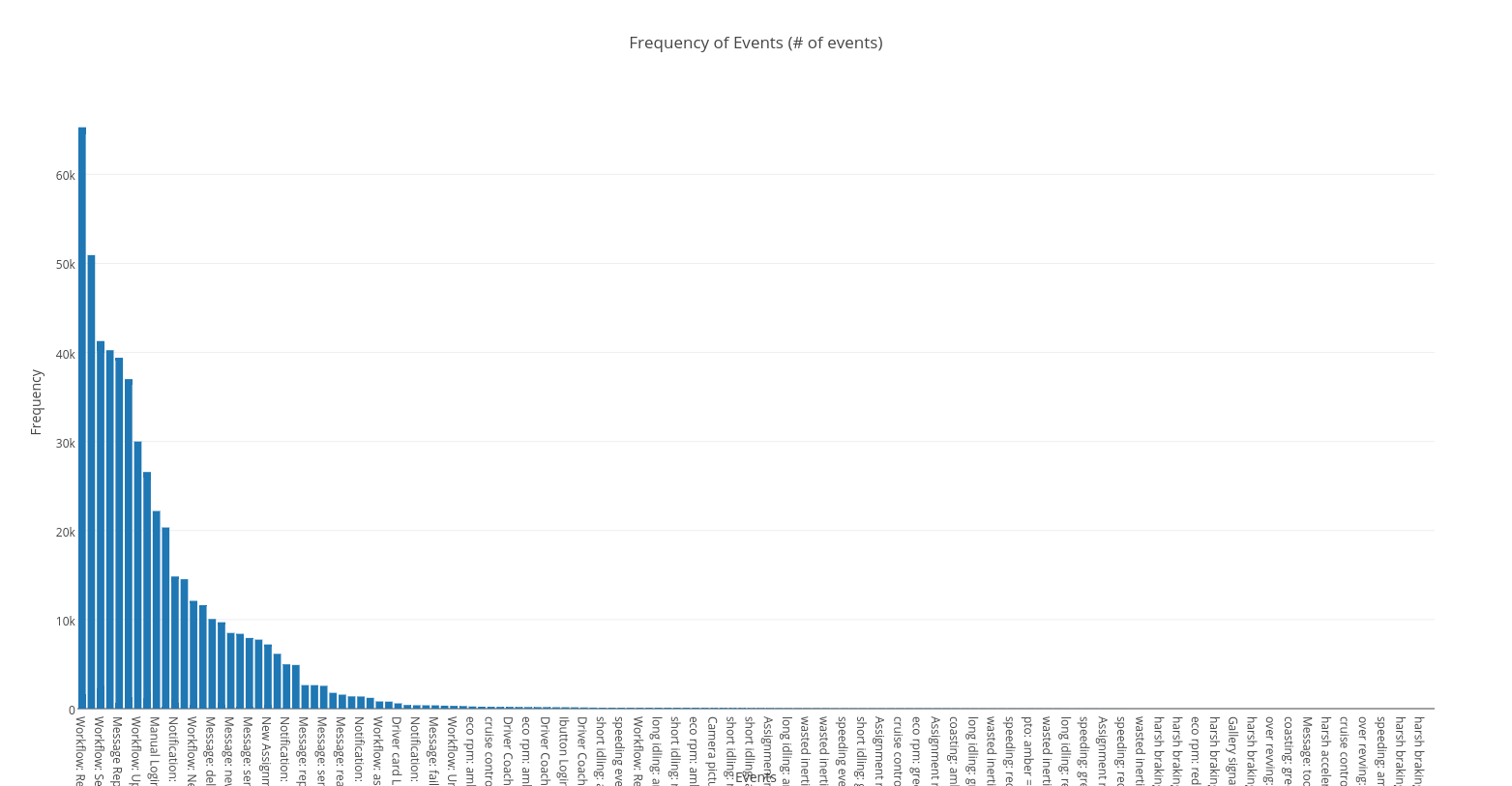


Figure 2 Frequency of Events

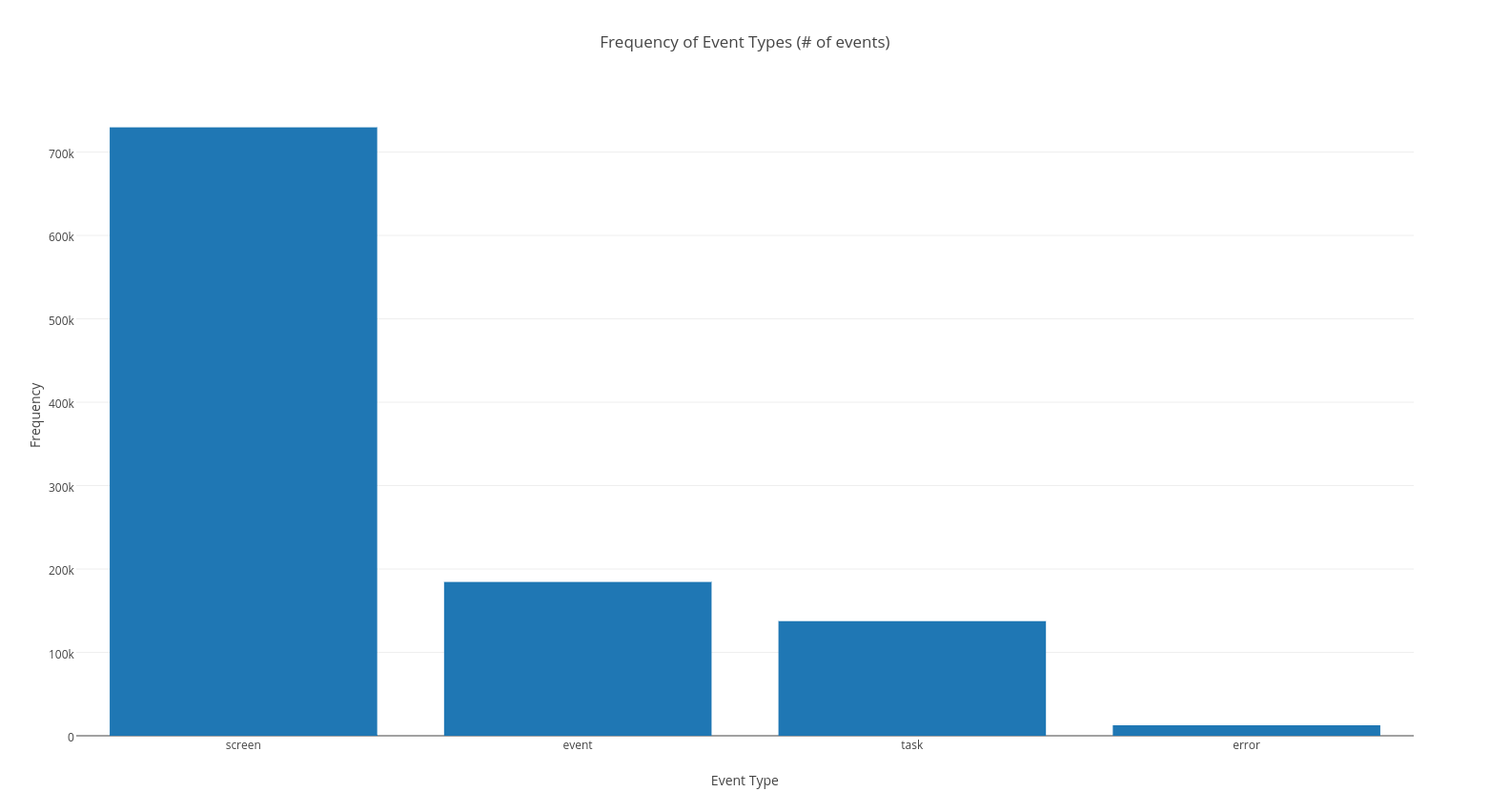


Figure 3 Frequency of Event Types

# Goals

At the core of UXprobe is idea of helping project managers figure out how to make their applications better. Using data from how the user uses an application to how satisfied they are with the features UXprobe tries to identify the strengths and weaknesses. Features of an application that have a lot of use must be maintained, if this feature also has bad satisfaction major improvements should be made. Before UXprobe project managers could only guess what is good and what is bad. UXprobe takes into account real life data from the people using the applications on a day to day bases. Therefore, this is the core of the data visualization. How can we visualize the way people use an application in way that tells us how to improve it? Is there something hidden in the data that can only be discovered through visualization?

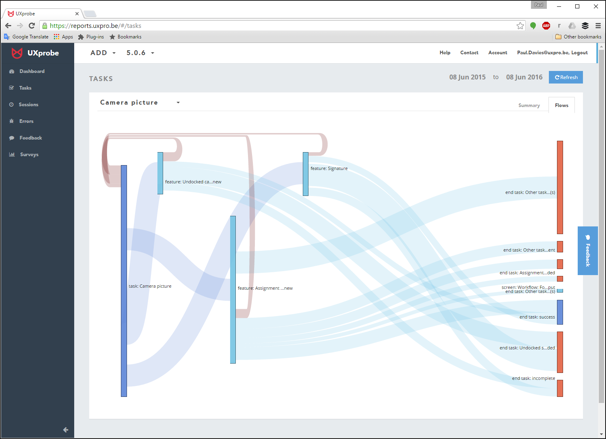
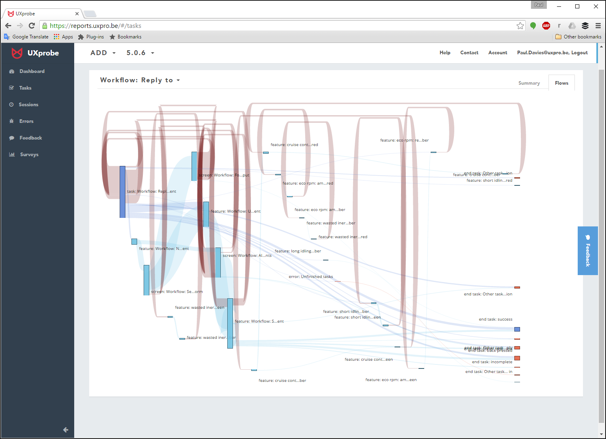
# Task

UXprobe provides their customers with a dashboard that shows them the use patterns in their apps. Using standard pie and bar charts they can get an idea of how people use their application. Recently UXprobe has attempted to implement a Sankey diagram to help get a global perspective. The system is too big for this type of visualization; it makes it very difficult for a practical implementation. The task is to have a data visualization that can capture global trends while also retaining local clarity. This will allow project managers to really get an idea about what is going on with their applications. In the case of the example data we have received that application is very specialized, therefore the project manager will most likely never use the application. This makes it relatively difficult to make design decisions.

# Existing designs

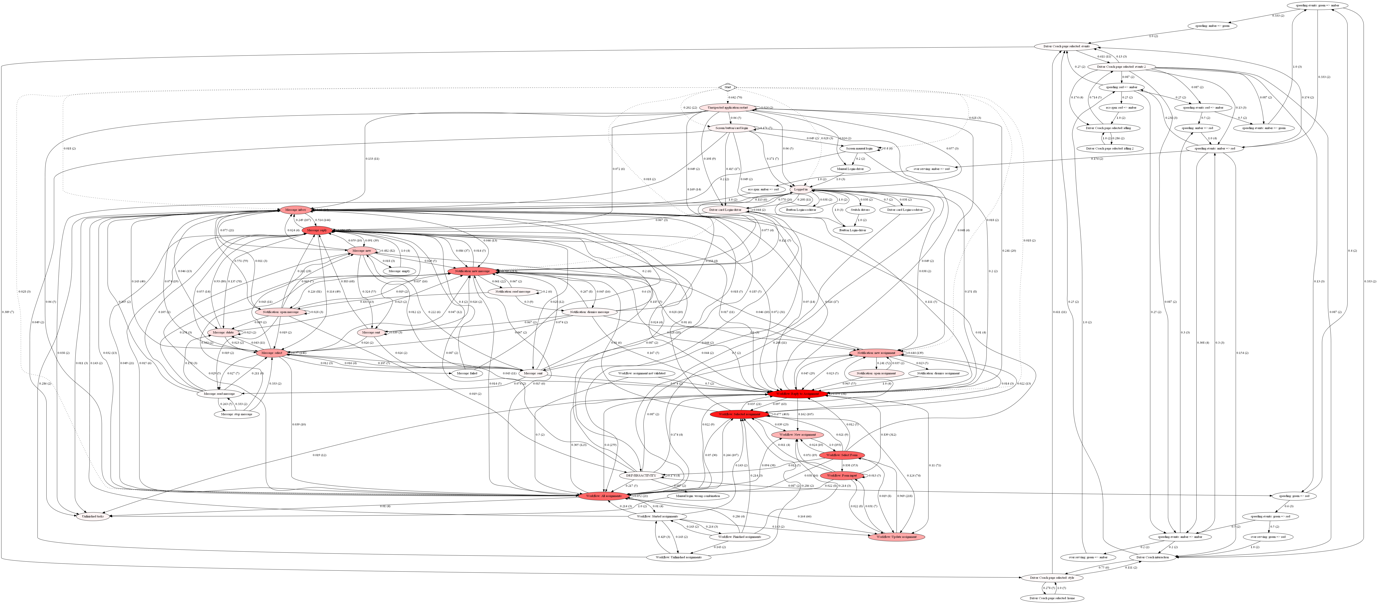
## Sankey diagram (uxProbe)

The dashboard of uxProbe provides an elaborate representation of flows between events within a task. It is a Sankey diagram, improved with reverse loops. This design works well when there is a natural flow and progression through different steps of a process, and shows also when the user goes back to a previous event in a process. However, when the user can arbitrarily go from one screen or feature to the other in a task, or when there is no natural order (think choosing tabs in a tabbed window), we start having a hairball effect due to the multiple feedback loops. The concept of flow might not be appropriate in this case. The presence of infrequent but legal sequences in the flow also clogs the design. Adequate labelling becomes a challenge.

## Graph diagram

A group of students from a computer science program (Christof Vermeersch, Joachim Schreurs, Servaas Vandecappelle, material obtained from uxProbe), worked on graph representations of the succession of events, which delivers the type of output seen in figure XXX. They applied clustering techniques to reduce the complexity of the graphs and produced several types of output, but nevertheless, we see here the limits of using graph representation for large networks with many loops. Other than some complexity perception based on the density of the hairball, there is no emergent perception about the use of the application.



# Design Space

After combing through the data we came up with a handful of initial designs. In reality there is only so much information contained in the data. Brainstorming kept on leading us back to only two types of perspectives, a network perspective, which is how tasks or events are connected. The second is a flow perspective where you can see how a user moves throughout the task.

Figure 4 shows one of the network perspectives. The double circle indicates a starting event. A green circle indicates the end of a successful task and the red the end of an unsuccessful task. Successful is defined as task/event which is completed. Unsuccessful is defined as a task/event which has failed due to error or abandoned by the user. The colors in the ring describe various levels of user satisfaction. This was not chosen due to scaling issues. There are too many tasks, therefore there would be no way to get a view of the system as a whole. Drill down options were considered but it was decided that a global outlook was better.

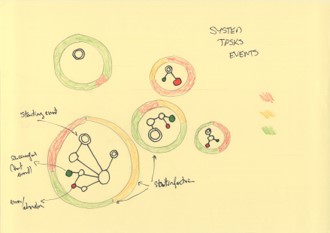


Figure 4 Network Bubble Chart

Figure 5 represents another network perspective of our data, this time using location of the task network as a function of use and satisfaction. Unfortunately, the satisfaction data was too scarce to implement this design. The idea was to have a network of events in a task. If this network was in the top right corner, it has high use and high satisfaction, therefore the project manager shouldn’t change anything. If it was in the lower left hand corner, it has low use and low satisfaction, therefore he should consider removing this feature. High use and low satisfaction task would have needed improvement.

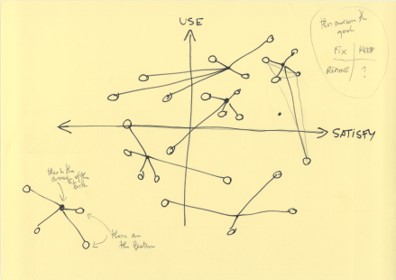
Now we switch to the flow perspective. Figure 6 explored how the user flows through the application. It is a discrete parallel coordinates graph. The bins represent the event, while the size of the bin represents the frequency. The color of the line show whether or not an even is successful. This design suffers a few of the issues mentioned about figure 4. Scaling would be very difficult. The more data there is the less readable the design is. This also looks at the system at a more abstract level, giving the user no option of looking at the event level.

Figure 5 Network Graph

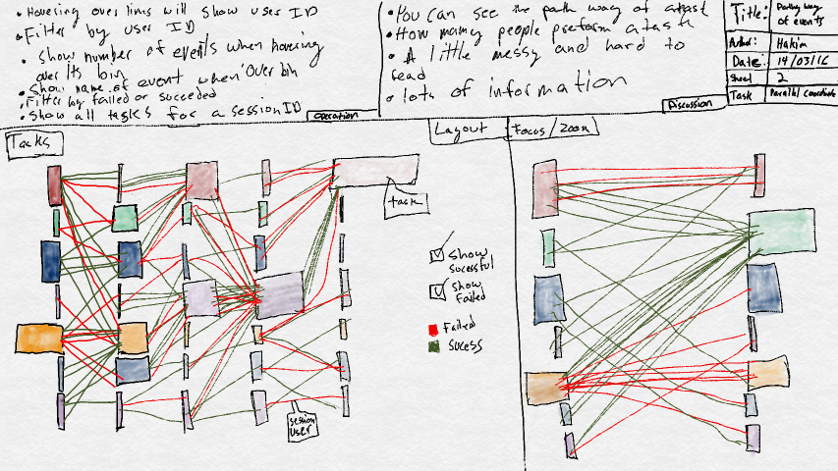
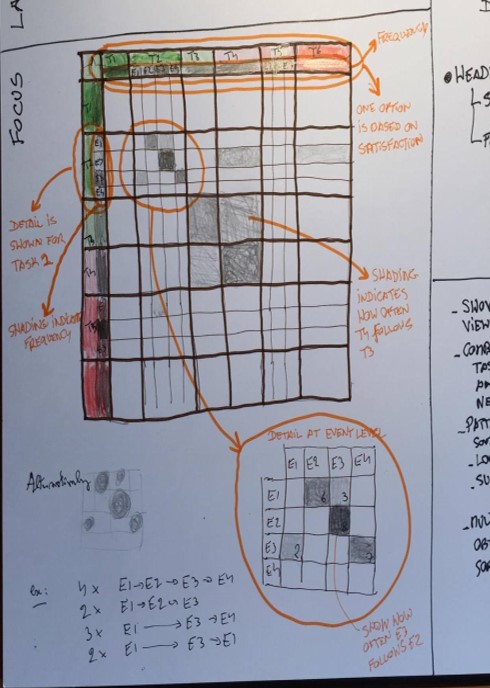


Figure 6 Parallel Coordinates



The final design was an adjacency matrix. This design allowed us to view the system as a whole, as well as at the even level. It shows flow (at an event level) as well as a network. In figure 7 you can see how look at this matrix you can tell if an even follows an event and the frequency. The original idea was to have the matrix so flow at the task level and then to expand when clicked on. In the end this did not pan out. The border of the matrix shows satisfaction data, but due to the sparseness of this data it was not included in the final design.

Figure 7 Adjacency Matrix

# The final design: a tiled adjacency matrix

## Principles

The adjacency matrix represents the network of events, where the vertices are events (screens, features, errors) and the directed edge values correspond to the frequencies of an event following another event. Two fictive events are added: an event “Start” to indicate the beginning of a session, and an event “End” to indicate the end of a session. All events no associated with a task were allocated to a fictive task “No Task”.

The labels of events start with a one letter code, F for feature, S for screen, E for error. These could be replaced by glyphs.

Events are grouped by tasks. Each task has a heading with it’s task name, and an empty row/column separates the cells of different tasks in the body of the matrix (we call these the task tiles).

The ranking of frequency of tasks and events is represented by their position: tasks and events to the top/left, are more frequent. Complexity of tasks (their number of different events) can be inferred from the size of the columns/rows or by the area of the task tile (on the diagonal of the matrix).

Errors are a particularly important event to single out, so we represent them with a red hue. Since the events are sorted by decreasing frequency, the position of a red label among the other labels of a task makes it easy to spot if errors are more or less frequent within a task.

The color content of a cell (i, j) of the matrix corresponds to the frequency of event-of-column j following an event-of-row i. The relative frequency is represented with color intensity (in P5 we implemented this with a degree of transparency on a white, background as P5 uses RGB coding). We saw that in our dataset the sorted frequencies of events are exponentially decreasing, so we used a formula to map the frequencies to a fixed interval (which is the base for the color intensity) after a log transformation. If the frequency is 0, then the cell is white.

The cells in columns of error events, indicative of events followed by an error event, are colored in red. Error rows are not in red, because we don’t want to particularly focus on normal events that follow en error.

A slider called “contrast”, increases the color intensity to a maximum for all non-white cells. The aim is to provide the user with a view on what sequences of events are possible in the system, no matter how little frequent.

Another slider called “filter”, cuts out the lower frequencies, turning the cells to white. This filter allows the user to focus on the most frequent patterns of use.

A tooltip is shown when hovering over the matrix cells and gives details about the the cell, such as the two events involved, a measure of frequency.

## Technical implementation

The implementation consists of multiple stages a preprocessing stage, a processing stage, and a visualization stage.

* Preprocessing stage: data loading, recoding, cleaning, volume reduction, building data model and relationships. This involved designing among others Dataset, Model, Session, User, Task, Event, and LogEvent objects, with all the properties and methods needed to store the data, build their relationships, and implement the algorithms.
* Processing stage: mainly a Matrix object responsible for preparing the adjacency matrix, implementing the sorting, counting, value transformation algorithms, generating the interface files (CSV format) for the visualization in the last stage.
* The visualization stage: loading the interface files with matrix values, labels etc. and the actual drawing algorithms and interactions in a web browser.

## Technologies used

* The preprocessing and the processing were implemented in Swift 2.2, a recent language originally developed by Apple. First released in 2014 it became open source in 2015 (<https://swift.org)> .
* The visualization was implemented in JavaScript and the library p5.js (<http://p5js.org)>.
* File management was supported with shell scripts
* Collaboration was supported with Dropbox ([www.dropbox.com)](http://www.dropbox.com))
* R and RStudio were used for some initial statistical exploration. <https://www.r-project.org>.

## Data quality issues

A few problems caused issues in the implementation.

* The sequence of events is deduced from the timestamp. Unfortunately, the time stamp in the data set was precise up to seconds, which is a too coarse granularity. This made it impossible to determine actual sequence of many events because several events could appear to have taken place at the same time. We could not work around this issue. Among other issues, this could cause the matrix to appear more symmetric than it is in reality.
* In our second prototype, when we implemented the tasks grouping of events, there was a large number of events appearing repeatedly in all tasks. This was breaking the concept of our adjacency matrix. It turned out these were some sort of system events. Typically, if the driver drives too fast, an indicator in the application for speed control will turn from amber to red for instance, and this generates an event ‘speeding amber to red’. These events are not part of a task the user is performing, but in the data set they were allocated to whatever task the user was performing at the same moment. As a workaround we identified some commonly occurring characters in the name of those events and programmatically allocated them to a fictive task “System Events”, which could then be excluded from the processing.
* In our prototype version, besides the system events, there were also a large number of events apparently allocated to different tasks. This also broke the logic of our design (based on the concept of events of a task being specific for that task, with a n-to-one relation), and increased the size of the matrix. Investigating these cases, it turned out that they were non-sensical combinations of tasks and events, and their frequency less than 10 occurrences in a million records. This provided an exclusion criterion.
* There is a substantial number of events falling into the category “No Task”. Quite a few of them seem to be typical of existing tasks (ex: “Message: send”), but we didn’t try to correct this, the design takes care of it and allows the user to identify issues in its logging system.
* We noticed that the different part of the data set came from different versions of the system and that some events might represent the same thing but might have changed name. Could not workaround this issue. An evidence of this can be found in two events almost identical, but with “picture” spelled “picure” in some cases.
* Tasks are only identified by their name; they do not have a unique identifier for each instance. Therefore, if multiple events come after each other with the same task it would be impossible to distinguish if it was one or multiple tasks. In the current version of the design, this doesn’t create problems, but if we aggregate at task level, it wouldn’t work anymore.

# The actual matrix: realization and evaluation



The resulting matrix is shown in figure XXX. What do we see and how does that answers the goals of the user?

* The most frequent task is the no task category, and we see some very frequent high frequency error ‘Unexpected application restart’. UXprobe explained that their reporting allowed the customer to detect an error in their application which triggered constant restarts. The matrix clearly clearly identifies the issue and makes it very “visual” to perceive.
* We get a feeling of the overall complexity of the application. Leaving aside the “No Task” task, we see that the most complex tasks are also the most frequent, namely “Workflow: Reply to assignment” and “Message Reply”. Less complex tasks are also not frequent, such as those related to the camera of the tablet.
* There are some “pillars” visible on the matrix, these represent events that are likely to follow almost any of the other events. Without surprise these are the screen “Workflow: All Assignments” and the screen “Message: Inbox”
* Equally remarkable are almost empty lines and columns with high frequency. It’s the same screen “Screen button/card login” that appear in all tasks related to assignments and and messages. We should ask the user to get an explanation for this.
* A recognizable pattern is a plain square. We have one in “Workflow: Reply to assignment”. A square involving a group of events signals that any event from the group can be followed by any of the other. They are contiguous in terms of frequency. We can compare this with the Sankey diagram of UXprobe, where we saw many backward loops. It means there is no natural linear flow from a start to an end of the task.
* A very interesting pattern is the one of task “Driver Coach Interaction”. The driver coach is a part of the application giving feedback on the driving style of the driver. All events are screens. Clearly from the “square” pattern, we can infer that the screens are accessed in a more or less random order. We also see that all screens must be visited with a fairly same frequency. However, looking more in detail, we distinguish a pattern in the pattern: two diagonals appear to have a higher intensity, one above and one below the main diagonal of the matrix. Let’s use the contrast and filter sliders to isolate these patterns. These two diagonals are what we had in mind when we did the first 5-design sheet of the matrix design. The diagonal just above the main diagonal of the main diagonal means that the screens are visited one after the other, in the same order. The diagonal below the main diagonal show the reverse: the driver selects the screen in the inverse but still linear order. To conclude about the pattern of use of “Driver Coach Interaction”, it seems that the different screens are accessed in a random order, but that more often, the driver will visit them in sequence in one direction, or in the other direction.



Let us remind that there are some serious data quality issues, and that we don’t know the application. However, as outsiders, we were nevertheless able to get an overall feeling of how the application is used and to emit some hypotheses about some specific patterns of use.

# Are the goals fulfilled?

As a user, namely a project manager, that wants to visualize how the application is used in order to prioritize his actions, we believe the matrix design offers some substantial benefits.

* First of all, it shows the entire application.
  + Though the size of the matrix will grow with the number of events, the complexity of sequences and relations between tasks and/or events remains entirely contained in the matrix. There won’t be any hairball effect. And if the size of the matrix becomes an issue, collapsing tasks as planned in our original design can easily solve the issue without removing significant information. (if it does create an issue, in case of a very large numbers of events with significant interaction across numerous tasks, the question is: isn’t there an issue with the structure of the application in the first place?).
  + This is an advantage compared to the Sankey design that is limited in scope to a single task.
  + Compared to the graph diagram, we avoid the hairball effect
* The complexity of the system and the most frequently used tasks and events are immediately perceived. The project manager can assess whether the complexity of tasks is consistent with the frequency, and the overall strategy of the system. Maybe the most frequent tasks are the core business and require elaborate features, while less common and less critical tasks should be kept simple (also because money should not be spent there). But maybe frequent tasks should be simpler because they need to be fast to operate. Most software for a wider public such as those used in our smartphone choose that option.
* Within tasks, the relative importance of use of the events is equally important.
* Some significant patterns of use are immediately identified:
  + pillars indicating some major events that need to be immediately accessible from everywhere in the application. The project manager can compare this with the current design of the application: does it facilitate this immediate access? Should the application be modified?
  + empty lines signal some particular event or some issue
  + plain squares indicate some events that can happen in a random order
  + diagonals can indicate a linear flow of events
  + contrast and filters allow to isolate some patterns, as we have shown, where both random order and sequence order where present at the same time.
* Errors are major negative events for the user. The matrix design allows an instantaneous identification of errors, their frequency, and their localization or cause in the application structure.
* The matrix can be miniaturized and still give a overall perception of the use of the applications. This could allow comparisons of different uses of the same system, or different implementations.
* In general, patterns of use can be compared to the current version of the application and the project manager can assess whether the application correctly supports the use patterns, or detect some patterns of use dictated by the application.

# Further improvements

* Collapsible tasks
* Frequency histograms for the tasks
* A master/detail design, where additional information would be displayed in a second design for a particular cell (an more elaborate version of the tooltip)
* Visual aids to easily locate the labels, such are guidelines, highlighting or magnifying the labels corresponding to a cell
* Different sorts of sorting.
* Use of satisfaction results of the micro-surveys

## First prototype???

The first prototype of the design can be seen in figure 8. The first row is the start of a task and last row is the end. The squares show how often events follow each other.



Figure 7 Prototype 1