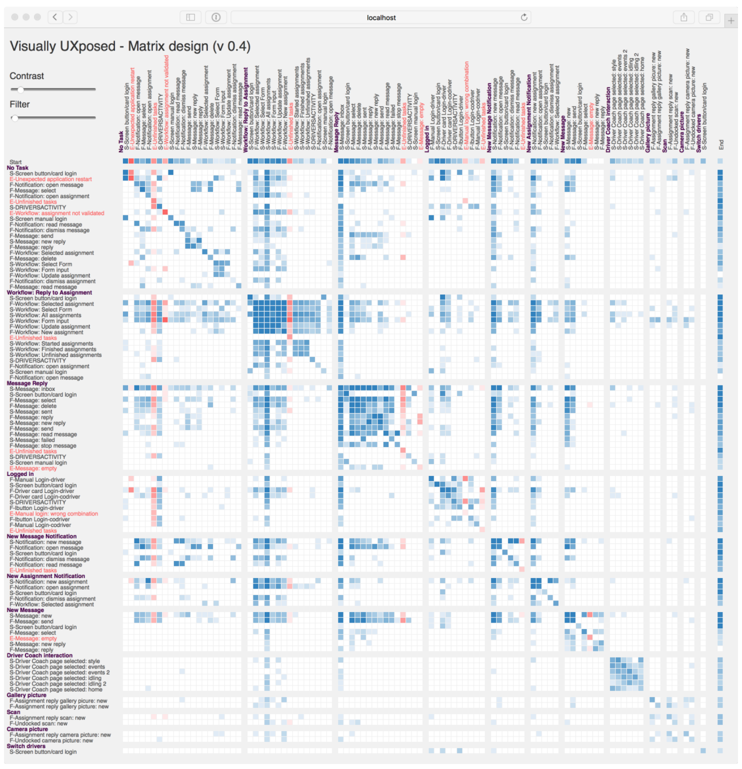
Visually UXposed



June 9th, 2016

Ivan Colling
Mohamedhakim Elakhrass

KULeuven | Capita Selecta : Visual Data Analysis | 2016

Introduction

The purpose of this project is to develop an original design for the visual data analysis of a dataset brought by a domain expert. We chose the project presented by a start-up developing expertise and providing solutions for user experience measurement.

The domain expert and UXprobe

Our domain expert is Paul Davies, one of the two founders of UXprobe (http://www.uxpro.be/). This startup targets IT managers and offers services that allow them to improve their application. To achieve this, UXprobe systems track how users interact with the software of their client. The objective is to assess how successful the user is at completing tasks in the application. At the same time, satisfaction is measured right at the moment the user performs an operation. The unique value proposition of UXprobe is to measure success and satisfaction in context. Customers of UXprobe have access to a dashboard with metrics of use and satisfaction. They can drill down to any operation of a particular user, for a particular session with the software.

The dataset

The source

The dataset we are working on is a sample of data provided by UXprobe about one of their customers. This customer offers fleet management for transport companies. Truck drivers use a tablet that runs a software developed in-house. This system is a navigation system (TOM TOM), but also handles communication between drivers and the company, manages the transport assignments, supports time management and driving style coaching.

Initially, the software was complex, offered many features and was difficult to operate for the final user (the driver). UXprobe allowed to analyze the way the users interact with the application. Their systems record each operation of the driver with the application. The level of satisfaction of the user is collected by prompting micro-surveys and feedback questions. These micro-surveys are tied with a particular operation in the application, so that the satisfaction can be assessed in the context of a particular use, for a particular user.

Information about use and satisfaction with the system allowed the fleet management company to adapt their software to the needs of the actual users, the truck drivers.

“Use” data

Our first data set contains about a million of log entries. Each log entry contains: a session identifier, a user identifier, a task name, an event name, a timestamp. When a user starts interacting with the software, he enters a *session*. A session can last for several hours, typically 7 or 8 hours (corresponding to the limitations of the time a driver is allowed to drive a truck before resting). During that session, the user will want to achieve several actions, such as checking his assignments for the day or sending a message for instance. Such actions are called *tasks*. To perform a task, the user will interact with the system, and perform some operations, these are called *events*. Events fall into different categories: a *screen* (a screen pops up), a *feature* (some menu or action done by the user), an *error* (error message). During the session, several actions can be initiated by the user, with a variable number of events for each task.

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 the logging system. In the beginning the granularity was coarse, with precision up to seconds. This means that events could appear to be timed at the same moment, which can lead to some abnormal sequence of events when sorted.

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). This can be on purpose or could be the results of a data quality issue.

Volume of data

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

Description of the entities and their corresponding fields

Entity or field	Field name and description	Example of value
Session	The field name is *sessionID* and contains a unique identifier of session	`55df6ce6-bac0-4bbb-b7ab-2e824d8a01a4`
Record	The field *sequenceID* contains a unique identifier of the record, and can be ignored.	`3343ae5e-997d-11e4-bfb2-f7c03b184ea7`
Task	A field *taskID* does not contain an ID, but a short description of the task.	`New Message Notification`
Event	The field is called *activityID* but contains a short description of the event rather than an ID	`Notification: new message`
Event type	The field *type* contains the type of the event, which can take 4 values: ‘screen’, ‘event’ (where here it is ‘feature’ that is meant), ‘error’, ‘task’ (which is actually a fictive event to mark the beginning of a tasks.	`screen`
User	A field *userID* contains the identification of the user.	`VEACESLAV OK-000000000046Y000-327700050402453`
Start time	A field *starttime* contains the time that an event starts. In the dataset we received, the time has a maximum precision of seconds.	`2015-01-11 11:19:04`

Missing information and missing data

The structure of the software or the relationships between tasks and events are not available and will be inferred from the data. The actual sequence of events is not directly available in the dataset, and we inferred it from the timestamp: an event is followed by another event within a session if it has a timestamp that immediately follows. As already mentioned the time granularity was an issue. About 50% of the *userID* fields are empty. Around 450 thousand events do not have a task.

Statistics

These statistics show the distribution of tasks, events and even types from the raw dataset. They are measured in terms of records (occurrences of events). We see that sorted by decreasing frequency, they are exponentially decreasing.

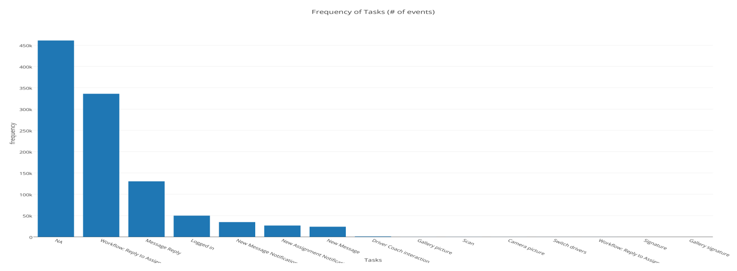


Figure 1 Frequency of Tasks

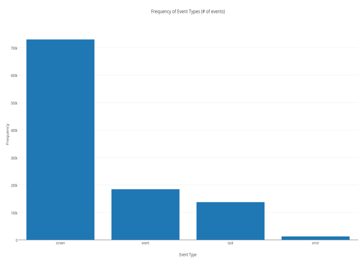


Figure 2 Frequency of Event Types

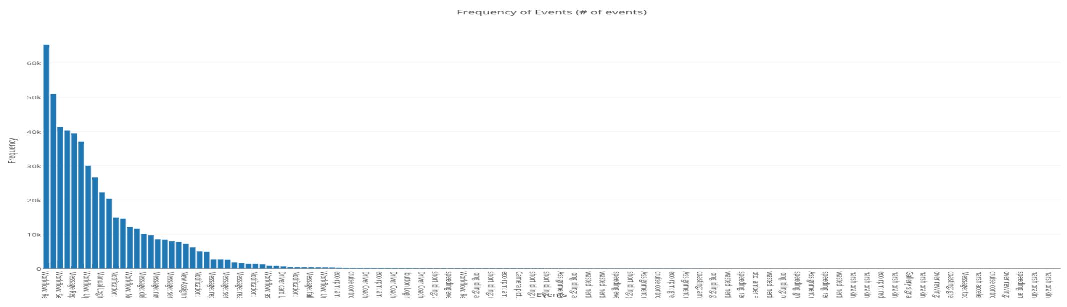


Figure 3 Frequency of Events

The user

We call final user the person that will use the visualization, so this would be a customer of UXprobe, typically an IT project manager.

Goals

To support the project manager decision process about application development and setting priorities about what features to add, remove or improve in an application.

Tasks

The project manager has to allocate time and resources and decide which change requests will be prioritized. He collects information about how the users interact with the software and their satisfaction. UXprobe provides support in that process by giving an actual and even real-time insight into the use of the application. Furthermore, analysis of success and micro-surveys are in-context tools to report satisfaction. UXprobe already implements some visualizations in their dashboard.

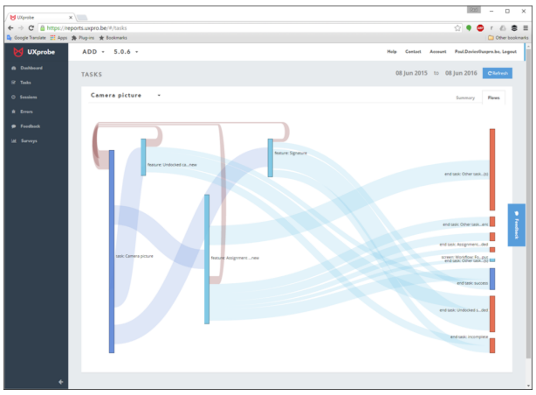
Our project objective

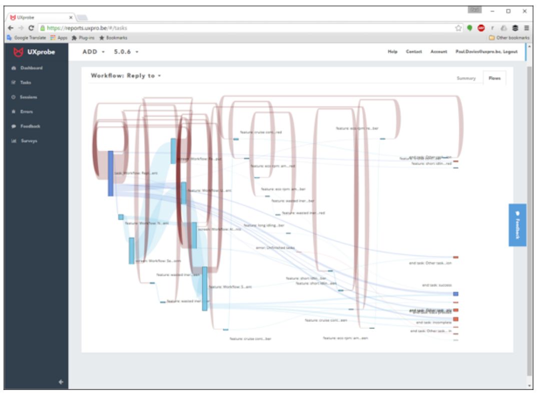
We will design and implement a novel data visualization to add to the tools UXprobe is currently offering to its customers.

Existing designs

Sankey diagram (uxProbe)

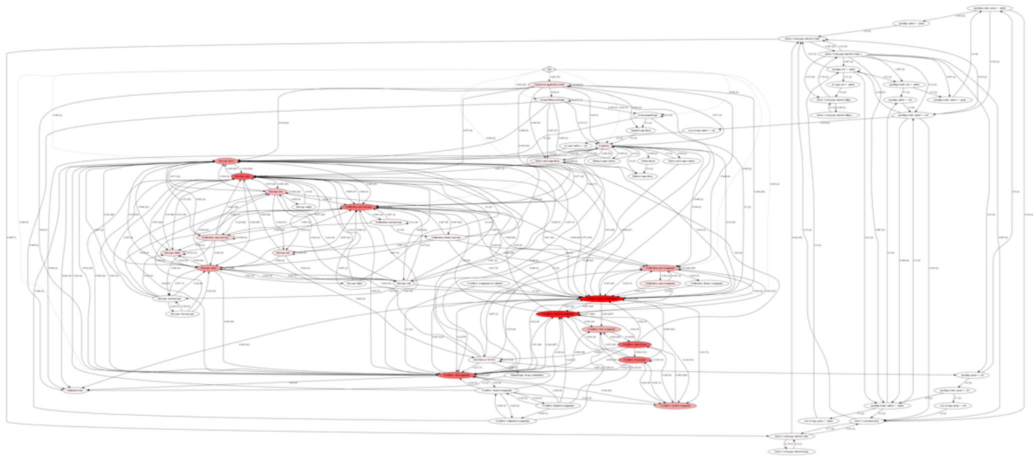
The dashboard of uxProbe provides an elaborate representation of flows between events within a task. It is a Sankey diagram, improved with backward flows to cater for loops. This design works well when there is a natural flow and progression through different steps of a process, with some options to go back to a previous step 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 presence of infrequent but legal sequences in the flow also clogs the design. Readable labelling becomes a challenge. Two examples are given below: tasks “Camera Picture” (left) and “Workflow: Reply to assignment” (right).





Graph diagram (students’ project)

A group of students from a computer science program (Ch. Vermeersch, J. Schreurs, S. Vandecappelle, material obtained from uxProbe), worked on graph representations of the succession of events, which delivers the type of output seen below. They applied clustering techniques to reduce the complexity and produce simpler graphs, but nevertheless, we see the limits of using graphs for large networks.

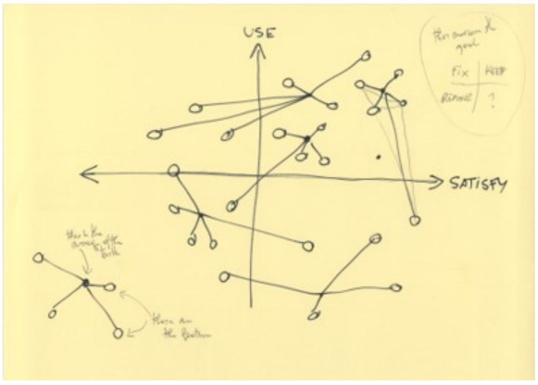


Exploring the design space

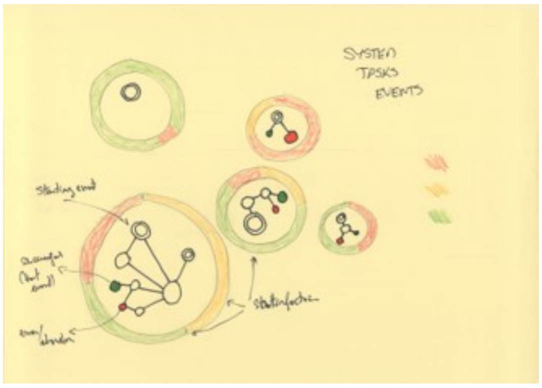
We used the 5 Design Sheets approach to generate ideas and designs.[1]

Early designs

The early designs focused on the two dimensions of use and satisfaction, and the hierarchical structure of the data (user -< sessions -< tasks -< events). One example worth mentioning is the one shown here. It had the ambition to provide a strategic decision matrix, with tasks and related events represented in a two-dimensional space of use and satisfaction. A task in the quadrant with high use and high satisfaction would require no change. A task in the quadrant with high use but low satisfaction, should require priority development, etc. But when we received the datasets we realized that there weren't enough satisfaction data for this option.



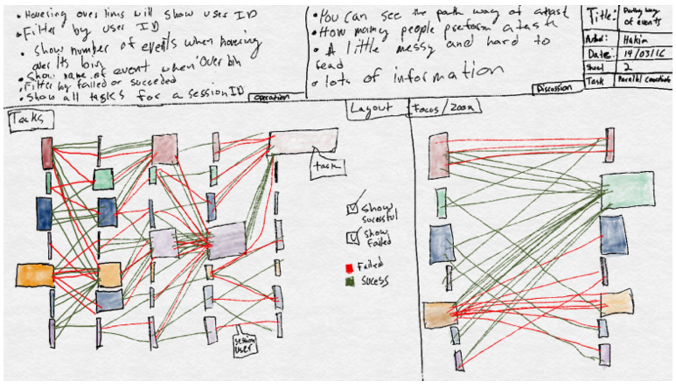
Design 1 – Network of events in tasks rings



Surfaces of cercles are proportional to frequency. The rings represent tasks; the colors of the ring represent levels of satisfaction (red: unsatisfied versus green: satisfied). Inside we have the network of events. The double circle indicates a starting event. A green disk indicates the end of a successful task and the red disk the unsuccessful end of a 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. This design would likely run into scaling problems, with conflicting ring sizes with space taken by events. Also a very challenging design to implement.

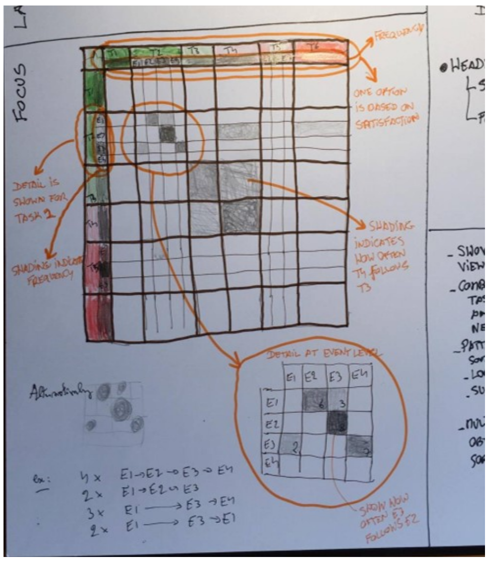
Design 2 – A flow type design perspective

The second design explored how the user flows through the application. It is a discrete parallel coordinates graph. The bins represent the events, while the size of the bin represents the frequency. The color of the line shows whether or not an event is successful. This design suffers some of the issues mentioned previously. Scaling would be very difficult. We wondered also how to integrate the perspective of events in relation to tasks. There is a succession of


• Hovering over lines will show user ID
• Filter by user ID
• show number of events when hovering over its bin
• Show name of event when over bin
• Filter by failed or succeeded
• Show all tasks for a session ID
Operation
• You can see the path way of atleast
• How many people preform a task
• A little messy and hard to read
• lots of information
Discussion
Title: Pathway of events
Author: Hakim
Date: 14/03/16
Sheet 2
Task Parallel coordinates
Tasks
Layout
Focus / Zoom
task
Show sucessful
Show failed
Failed
Sucess

[1] Roberts J, Headleand C, Ritsos PD. Sketching designs using the Five Design-Sheet methodology, IEEE Transactions on Visualization and Computer Graphics, 2015

tasks in sessions, but that would blur the visualization of succession of events in tasks. Also tasks boundaries are not clear in the dataset, several successive instances of a task can appear like one instance.



Design 3 – The adjacency matrix

The final design is an adjacency matrix. It represents the network of events. Events are the columns and rows, the cell tells how times the event of the row is followed by the event of the column. This design allowed us to view the system as a whole and it handles well the hierarchy between tasks and events. The original idea was to have the matrix presenting the flows at the task level and then to expand to events when clicked on. Tasks and events could be sorted according to several criteria, such are frequency (which was implemented) or satisfaction, or maybe a natural order of screens and features chosen by the application developer.

The final design: a tiled adjacency matrix

Description

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 not associated with a task were allocated to a task "No Task".

The labels of events start with a one letter code for the event type, 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 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).

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.

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 cells in columns of error events, indicative of events followed by an error event, are colored in red hue.

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 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, and a measure of frequency.

Technical implementation

The implementation consists of three stages:

- 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 processing were implemented in Swift 2.2, a language 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).
- R, RStudio were used for initial statistical exploration. https://www.r-project.org
- File management was supported with shell scripts
- Collaboration was supported with Dropbox (www.dropbox.com)
- Blog was maintained at https://vuxposed.wordpress.com .

Data quality issues

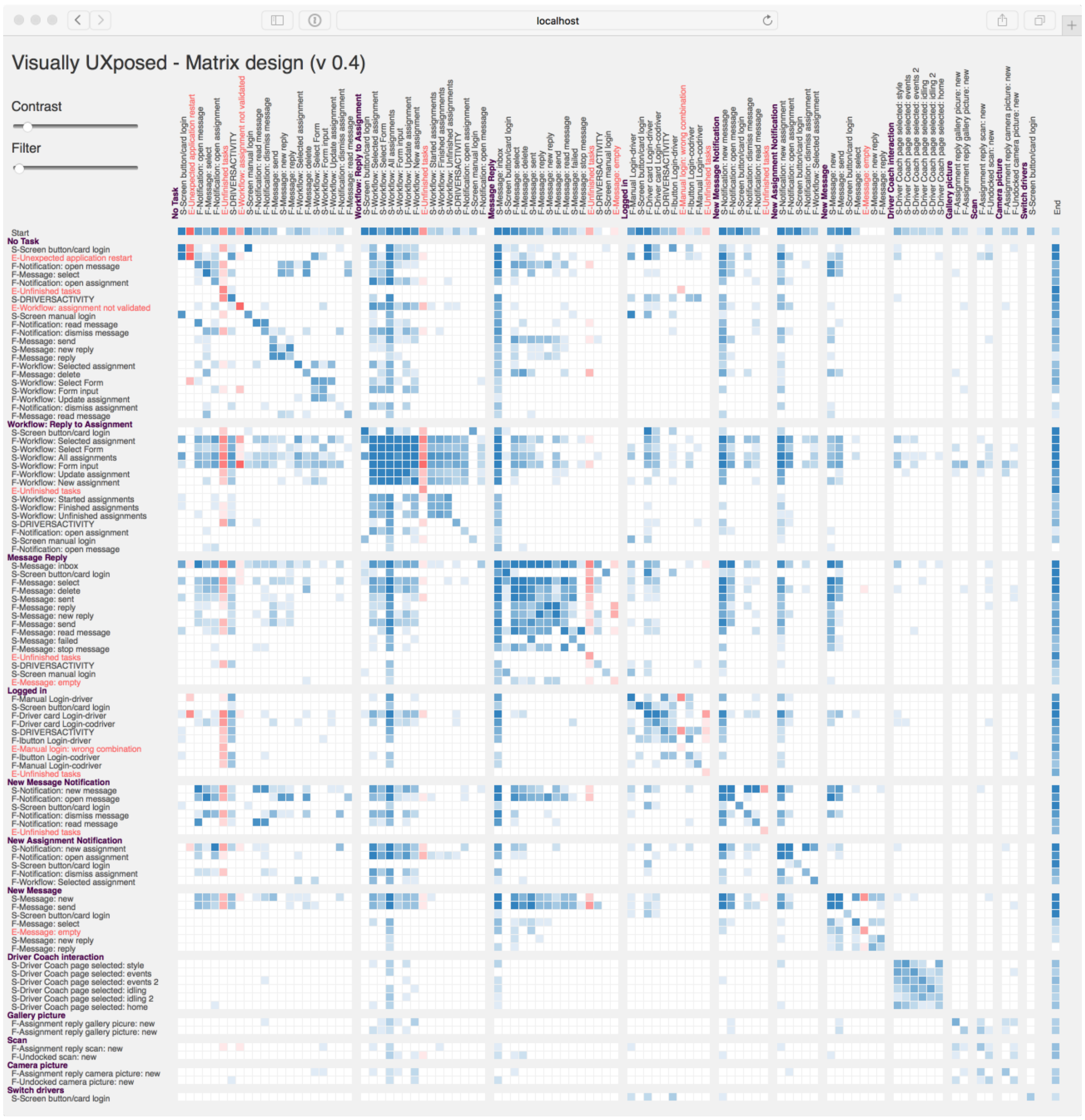
A few problems caused issues in the implementation, among the most significant:

- The time stamp in the data set was precise up to seconds, which is a too coarse granularity. This made it impossible to determine the actual sequence of many events because several events could appear to have taken place at the same time. Among other issues, this could cause the matrix to appear more symmetric than it is in reality.
- 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.
- A large number of events were apparently allocated to different tasks. This 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

relationship. 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.

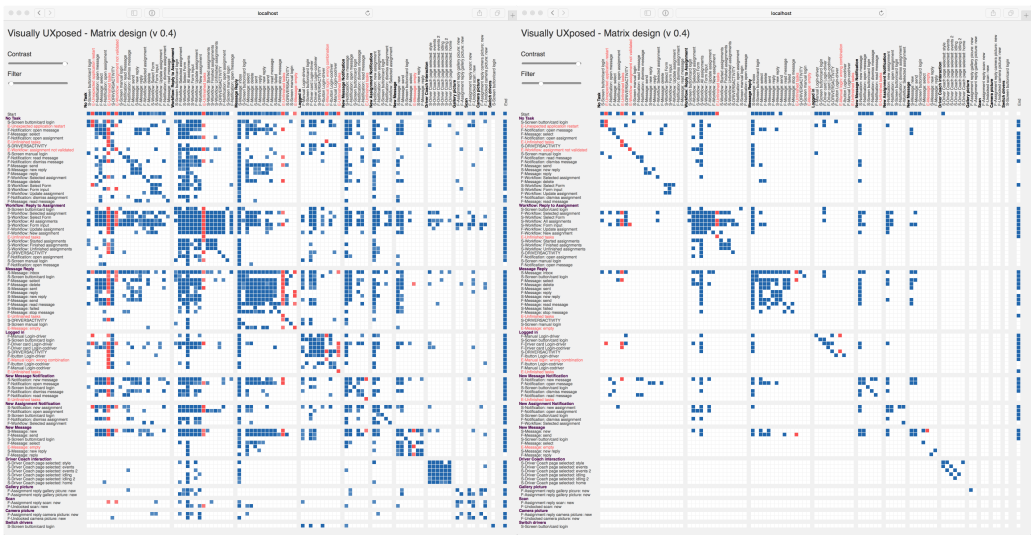
- A substantial number of events fall into the category "No Task". Quite a few of them seem to be typical of existing tasks (ex: "Message: send"), but we didn't try to correct this, the design takes care of it and allows the user to identify issues in its logging system.
- We noticed that the data set comes from different versions of the system. Some events might represent the same thing but might have changed name. We could not workaround this issue. An evidence of this can be found in two events sometime spelled "picture", sometimes "picure".

The actual matrix!


localhost
Visually UXposed - Matrix design (v 0.4)
Contrast
Filter
Start
No Task
S-Screen button/card login
E-Unexpected application restart
F-Notification: open message
F-Message: select
F-Notification: open assignment
E-Unfinished tasks
S-DRIVERSACTIVITY
E-Workflow: assignment not validated
S-Screen manual login
F-Notification: read message
F-Notification: dismiss message
F-Message: send
S-Message: new reply
F-Message: reply
F-Workflow: Selected assignment
F-Message: delete
S-Workflow: Select Form
S-Workflow: Form input
F-Workflow: Update assignment
F-Notification: dismiss assignment
F-Message: read message
Workflow: Reply to Assignment
S-Screen button/card login
F-Workflow: Selected assignment
S-Workflow: Select Form
S-Workflow: All assignments
S-Workflow: Form input
F-Workflow: Update assignment
F-Workflow: New assignment
E-Unfinished tasks
S-Workflow: Started assignments
S-Workflow: Finished assignments
S-Workflow: Unfinished assignments
S-DRIVERSACTIVITY
F-Notification: open assignment
S-Screen manual login
F-Notification: open message
Message Reply
S-Message: inbox
S-Screen button/card login
F-Message: select
F-Message: delete
S-Message: sent
F-Message: reply
S-Message: new reply
F-Message: send
F-Message: read message
S-Message: failed
F-Message: stop message
E-Unfinished tasks
S-DRIVERSACTIVITY
S-Screen manual login
E-Message: empty
Logged in
F-Manual Login-driver
S-Screen button/card login
F-Driver card Login-driver
F-Driver card Login-codriver
S-DRIVERSACTIVITY
F-Ibutton Login-driver
E-Manual login: wrong combination
F-Ibutton Login-codriver
F-Manual Login-codriver
E-Unfinished tasks
New Message Notification
S-Notification: new message
F-Notification: open message
S-Screen button/card login
F-Notification: dismiss message
F-Notification: read message
E-Unfinished tasks
New Assignment Notification
S-Notification: new assignment
F-Notification: open assignment
S-Screen button/card login
F-Notification: dismiss assignment
F-Workflow: Selected assignment
New Message
S-Message: new
F-Message: send
S-Screen button/card login
F-Message: select
E-Message: empty
S-Message: new reply
F-Message: reply
Driver Coach interaction
S-Driver Coach page selected: style
S-Driver Coach page selected: events
S-Driver Coach page selected: events 2
S-Driver Coach page selected: idling
S-Driver Coach page selected: idling 2
S-Driver Coach page selected: home
Gallery picture
F-Assignment reply gallery picure: new
F-Assignment reply gallery picture: new
Scan
F-Assignment reply scan: new
F-Undocked scan: new
Camera picture
F-Assignment reply camera picture: new
F-Undocked camera picture: new
Switch drivers
S-Screen button/card login
End

What do we see and how does that answers the goals of the user?

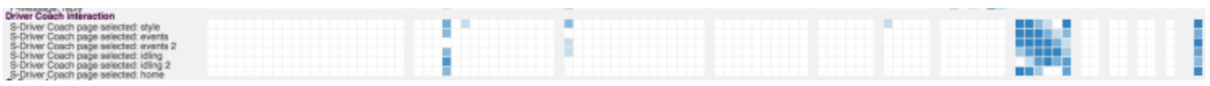
- The most frequent task is the “No Task” category, and we see some high frequency error “Unexpected application restart”. In one of our meetings, UXprobe explained that their reporting allowed the customer to detect an error in their application which triggered constant restarts. The matrix clearly identifies the issue and makes it very “visual” to perceive.
- We perceive 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 not frequent, such as the camera.
- 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 other of the group. 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.


localhost
Visually UXposed - Matrix design (v 0.4)
Contrast
Filter
localhost
Visually UXposed - Matrix design (v 0.4)
Contrast
Filter

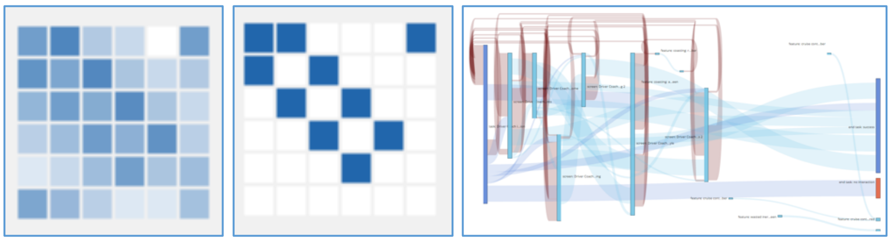
Matrices with maximal contrast (left), and some filtering (right)

Let’s look at a particular case: the “Drive Coach Interaction”.

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.



The task "Driver Coach Interaction" from the matrix (left), the same with contrast and filter (middle) and the Sankey version (UXprobe)

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 matrix 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, 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. We contrast this with the Sankey diagram of UXprobe.

Conclusion

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.

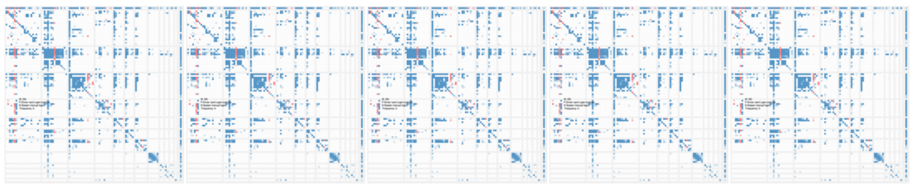
Are the goals fulfilled?

Strengths

As a user, namely a project manager, who 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. Although 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. It 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. 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. 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. And we can see from the patterns of the tiles how events occupy that space.
- 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?

    - empty lines signal some particular event or some issue
    - plain squares indicate events that follow each other 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 an overall perception of the use of the application. This could allow comparisons of different uses of the same system, after clustering, or comparing different groups of user, or different versions of the same system.



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.

Weaknesses

Patterns of use are represented in an abstract way, which allows to detect some emergent patterns, and cram the entire application space into a compact design. But for the layman, reading an adjacency matrix might not be straightforward, and a graph or a Sankey diagram, if the scope is limited in terms of number of vertices and edges, remains a better option. A solution would be to combine the matrix with a second visualization, in a master-detail fashion.

The current technical implementation uses a compiled language, Swift, to process the million records in a matter of seconds. The Swift program produces interfaces files in a way that the JavaScript part should only read the processed data and take care of the drawing. We experienced that the P5 implementation is rather slow, and at moments does not seem very robust. Interactions are difficult to implement and make the rendering even slower. So while P5 was a good option for prototyping, other technologies such as D3 would be better options.

Further improvements

Some ideas for improvement stem for the original 5-design sheet draft, others came along the way.

- Collapsible tasks
- Frequency histograms for the tasks (could be shown along the edges of the matrix)
- A master/detail design, where additional information would be displayed in a second design for a particular cell (a more elaborate version of the tooltip)
- Visual aids to easily locate the labels, such as guidelines, highlighting or magnifying the labels corresponding to a cell
- Different sorts of sorting.
- Use of satisfaction results of the micro-surveys, either as a sorting criterion, or surrounding the task tile, in the fashion of our first design. The green color is reserved for this purpose.
- Implement in D3