frontpage-wact.pdf

NUIssues: Optimising an issue tracker for touch to boost productivity

Martin Lehmann

8th May 2015

1 Introduction

Norman's (2010, 3) opinion piece 'Natural user interfaces are not natural' begins with the following quote from Steve Ballmer, CEO of Microsoft:

> I believe we will look back on 2010 as the year we expanded beyond the mouse and keyboard, and started incorporating more natural forms of interaction such as touch, speech, gestures, handwriting, and vision – what computer scientists call the 'NUI' or natural user interface."

Following the quote, a statement is made:

> [...] A new world of interaction is here: The rulebooks and guidelines are being rewritten, or at least, such is the claim. And the new interactions even have a new marketing name: natural, as in "Natural User Interface."
>
> As usual, marketing rhetoric is ahead of reality.

Norman is, of course, right: using Microsoft as an example, we have seen attempts at natural user interfaces in the direction of hybrid tablet computers like the Microsoft Surface, gesture trackers like Microsoft Kinect, and voice recognition in the personal assistant Cortana. Though efforts have certainly been made, we have mostly seen that Natural User Interfaces (NUIs) are just a new way to interact with the old (but sometimes slightly different-looking) interfaces we already know. A prime example how this application does not work without thought put into optimising for the task at hand is the "flat" look of Windows called "Metro" and then "Modern UI", which was supposed to fit *all* Microsoft devices (desktops, laptops, tablets, and smartphones).

Natural User Interfaces are, indeed, a new way to interact with Graphical User Interfaces (GUIs), and not something completely new. However, as we move away from pen and mouse and into the domain of touch

sensitivity, speech recognition, and several other "natural" interface types, we must also update the GUIs to suit the task at hand and the input type of choice.

For example, a tablet with a screen surface sensitive to both the natural interface of touch and the Tangible User Interface (TUI) (Ishii & Ullmer, 1997) of stylus (pen) should immediately be able to differentiate between the two input types, because a user normally only wants to use the stylus for tasks fine-grained than what can be done just as efficiently by touching the screen with fingers: for example, sketching or annotating a document.

When it has been established that NUIs are new ways of interacting with GUIs and thus get all of the benefits (and restrictions) of the GUI's exploratory nature, we must of course learn from the GUI design and how we have interacted with them before to best design natural interaction experiences – but looking to the past is far from enough. New guidelines must be written and revised before we can determine a common understanding of how natural user interfaces should be used. After all, Norman (2010, 3) calls them *not natural*.

This report looks at an issue (task) tracker designed with guidelines and best practises for Natural User Interfaces in mind. The board consists of three simple lists: **todo**, **doing**, and **done**. The user can interact with existing tasks and drag them between the lists.

The goal of the application and this report is to detemrmine how optimising a 2D interface for the natural interface type *touch* can help boost productivity in a simple issue tracking system.

2 NUIssues

In the previous section, we had a brief look at natural user interfaces as a concept and how it manifests itself in current technology, before presenting the problem this project attempts to answer: how can designing an issue tracker for the natural user interface type *touch* boost productivity?

This section is a presentation of the issue tracker, hereinafter referred to as *NUIssues*.

2.1 Definitions

Before entering a presentation of the issue tracking system NUIssues, a common domain language must be established.

- **Issue**: The conceptual, individual task to be completed. Represented graphically by *card*s, which are moved between *swimlane*s as the task's status switches between *todo*, *doing*, and *done*.
- **Card**: A graphical representation of an issue. To be moved between *swimlane*s as the status of the issue changes.

- **Swimlane**: A vertical container for issues to indicate their status. Each issue resides within a single swimlane. NUIssues has three swimlanes along the **x** axis of the GUI: *todo*, *doing*, and *done*.
- **Index**: Each *card*'s vertical (y axis) position in a *swimlane*, and (by convention) the relative priority of the issue.

2.2 Rationale behind building an issue tracker

An issue tracker in particular was selected because almost everyone has used a similar system before, be it through clinging Post-It!s to a wall or using digital systems like Trello, Atlassian JIRA, Asana, and Ding.io. This way, the user will already have a mental model for a simple issue tracker, and all focus can be on optimising the mechanics of the application for natural input. The primary goal has been to optimise the actual board interaction for **touch** input: the *natural* way of moving physical cards around is to touch and drag them.

Importantly, an issue tracker also lends itself to a highly interactive environment where almost everything can be moved to let the user customise the layout. This seems to be optimal conditions for a small proof of concept on touch optimisation.

From a software integration perspective, it is clear that visualising something as integral to an organisation as *tasks* in a clear way that provides quick insight into the state of a project will be very helpful. Given a supportive Application Programming Interface (API), should also be a relatively simple task to integrate the application with existing issue tracking systems in order to provide a more natural interface for managing tasks.

2.3 Scope and sketch

NUIssues is a system with a 2D GUI optimised for the Natural User Interface **touch**. As is to be expected of an issue tracking system, there are no aspects of Augmented Reality (AR), Virtual Reality (VR), or Mixed Reality (MR). These are certainly interesting areas to explore in the future, but not within the scope of this project.

Figure 1 shows a simple initial mockup for the application's main functionality. For comparison, figure 2 shows the implemented, much more refined and informative result.

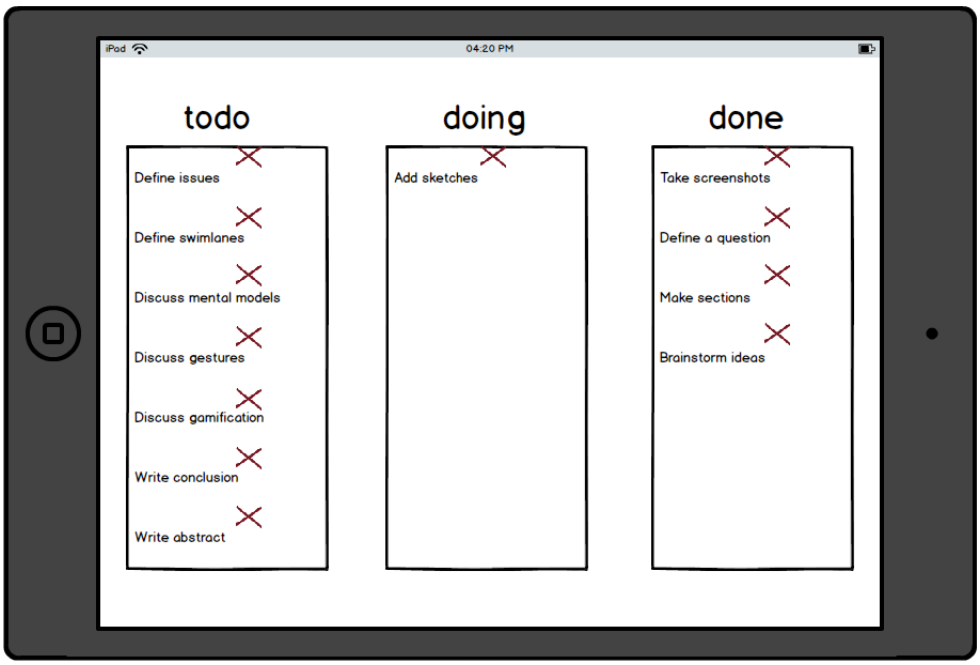

Figure 1: A simple mockup of the main functionality

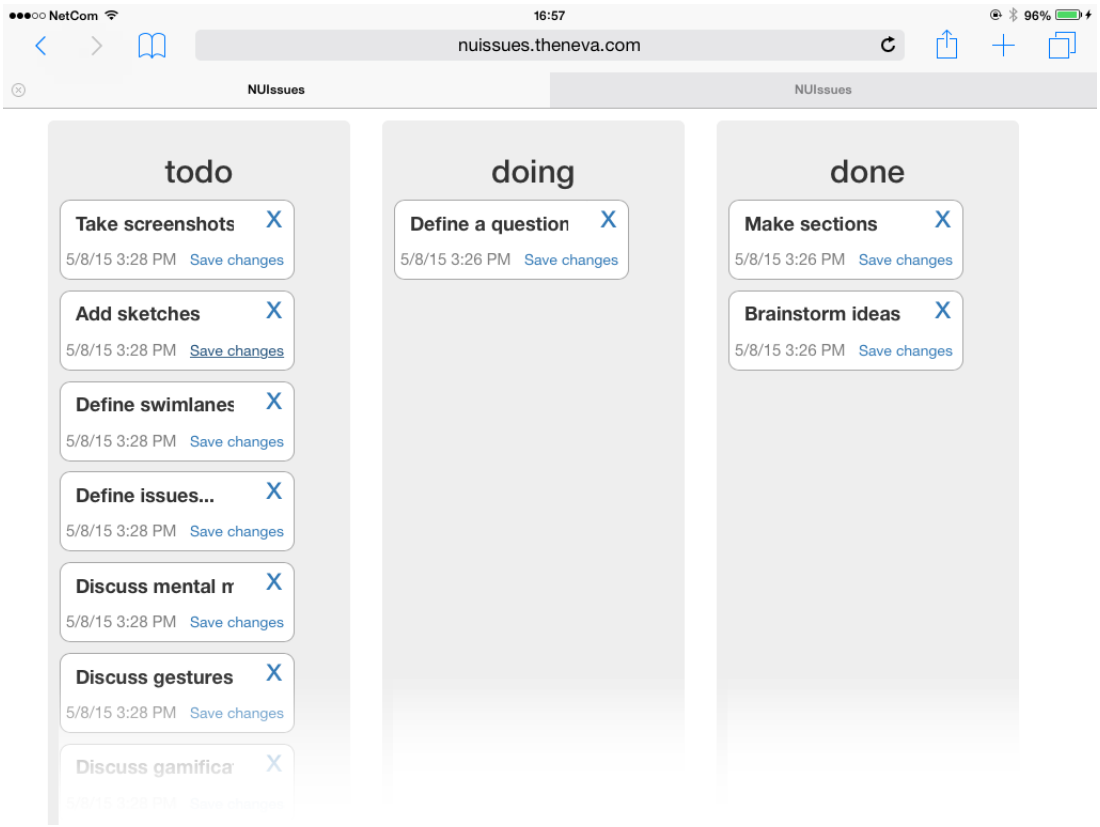

Figure 2: The system image for the main functionality

NUIssues **only** supports *touch* as natural input. While tracking changes on a user's skin and gaze tracking seem unrelated to the domain, there are several other to explore in future research: Gesture tracking, for instance, makes it possible to expand the application to run on interfaces like *touch walls*. Adding speech recognition or building a Brain Machine Interface can be relevant if the system is to be used by people who are physically incapable of interacting with a touch interface. Of course, all new interface type implementations add to the complexity of the application and makes it harder to keep the GUI optimised for a single type like touch. There are also obvious development costs related to expanding the software.

The interface is designed only for Apple iPad 1/2/Mini, but one may easily write responsive CSS to support more types of devices. This allows bringing the very simple interface to natural interfaces like *touch walls*, *touch tables*, in addition to smartphones and desktop computers. The interface is designed to be used when sitting down, although the application can certainly merely be opened if the user's intent is to grok the project's state while on the move.

If the user intends to edit textual content (only the title in NUIssues), it would be preferable to have an external keyboard. As reflected upon in the introduction, a keyboard is by far the most effective tool to write precise text in a computer system. An on-screen software keyboard will work almost as well, but overlaps almost all of the issues. Thus, the application is unusable for anything other than finishing text input and

saving changes when in the editing state.

The system is single-user, so inter-user task coupling (Wigdor & Wixon, 2011, p. 39) has not been explicitly addressed, but it is possible to evolve the system to support other forms of interfaces like touch walls or touch tables (or even just real-time updates for a project).

An issue that is not addressed in the proof of concept, but needs to be addressed in a full version of any issue tracking system is how one identifies users. In a touch-optimised system, it is preferable to avoid text input. Thus, one may look to how Apple identifies users on iPhone devices with fingerprint unlocks using a tangible user interface (TUI) (Ishii & Ullmer, 1997).

Any natural type of user recognition imposes hardware requirements on the devices that will run the application, but should be available as an option if they are viable for the organisation and target audience. Other natural forms of user identification include face recognition and speech recognition. A regular password option should be present, however.

Certain ethical concerns arise when identifying users. What can be stored on the device, what can be stored on a central server, and what should not be stored at all? Apple has already faced issues regarding their Touch ID system[1], and these are questions for further research to answer.

As far as gamification goes, there are no elements in the current implementation of NUIssues. However, applying the Mechanics, Dynamics, and Aesthetics (MDA) framework (Wigdor & Wixon, 2011, p. 107) to an issue tracker could be interesting, and very simple if one implements issue assignment to individual users. For example, the system could calculate points based on number of solved issues or their complexity, display a semi-permanent leaderboard, and give the users badges based on their performance and activities.

2.4 Mental models in NUIssues

Wilson and Rutherford (1989, 6) introduces the notion of *mental models* within Human-Computer Interaction (HCI), defined as any human's psychological representation (and implied expectations for) any given concept or object. A person will build a mental model for themselves, and for any object they interact with. They introduce a framework for the design phase of any computer system, with four distinctly different models at play.

First, one needs to define a *target system*. In this case, the target system is the NUIssues system and interface.

Second, there is a *conceptual model*, which is a very accurate psychological representation of exactly what the system does, usually held by an expert user or the designer themselves.

Fourth, the target system projects a *system image*, which is the implementation of the the conceptual model. This is the basis for the next element: the user's mental model of the system.

[1] http://venturebeat.com/2014/12/28/chaos-computer-club-claims-it-can-reproduce-fingerprints-from-peoples-public-photos/

Fifth, the *user* immediately builds a mental model of the system of the system based on the *system image*. This model is the basis for what the user expects the system to be able to do, mostly based on previous experience with similar systems but only based on the hints given by the application.

Last, the *designer has a mental model of the user's model*, in which the designer attempts to guess what the user's mental model of the system image will be like.

The last model is built on the target audience's expected background, and can be built by considering similar applications the user can be expected to have interacted with in the past. In this case, an obvious system is that of sticky notes and checklists to track tasks, which almost everyone has used at some point. Thus, these graphical representations of issues may be used as metaphors in the system. Further, the user can have come across several digital issue trackers optimised for web like [2], Atlassian JIRA[3], Ding[4], ServiceNow[5], and IBM Notes[6]. Thus, using common, good metaphors and following best practises is important when designing a system for use (which is, indeed, *any* system).

2.5 Behaviour models

MacKenzie (2003) introduces the Hick-Hyman Law which considers (average) response time when dealing with multi-option menus, and the Keystroke-Level Model (KLM), which considers with the user's time to actually complete their task. These are not considered further as NUIssues is a very simple application with no menus and only two core functions that build on existing knowledge from using the tablet that runs the application. One relevant aspect from the KLM, however, is the system response time factor: if the application runs on a slow physical server, the entire system will feel sluggish and slow to the user.

MacKenzie (2003, p. 4) also introduces Buxton's *3-state-model* for graphical input devices. While Buxton's model was defined for mouse interaction, the same "syntax" can be used to describe touch input. For example, in the scenario of dragging a card, the system has three states:

1. State 0: No contact with input (screen) (idle). Enables entering state 1.
2. State 1: Contact with card and moving (dragging). Lifting the finger off the card releases the card and sends the system back to state 0.

2.6 Development process

NUIssues is developed as a web application with touch support, designed for an iPad 1/2/Mini. The question this project attempts to answer (how can optimising an issue tracker for touch boost productivity?) is not

[2] https://trello.com
[3] https://jira.atlassian.com/secure/Dashboard.jspa
[4] https://ding.io
[5] http://www.servicenow.com/
[6] http://www-03.ibm.com/software/products/en/ibmnotes

(very) concerned with different device sizes for the same device *type*, so this was a simple way to prove the concept without complicating matters. Thus, the code base is rather clear – but also very focused.

Of course, building a native application would give access to higher performance and complete control of the environment. However, native code is often more complicated and can not be easily expanded to support more than one operating system. Unlike native applications, web applications with mobile support can be run on all modern device types, including phones, tablets, and web browsers and can potentially also be used on a touch wall or touch table interface as long as the software can run a web browser that can execute modern JavaScript.

No Software Development Kits (SDKs) specific to Natural User Interface design have been used, although the application leans on several frameworks and libraries commonly used in the MEAN (MongoDB, Express, AngularJS, Node.js) software stack:

- AngularJS is used as a front-end Model-View-Controller (MVC) framework with support for communicating efficiently with a server,
- ng-sortable is an AngularJS module for building interactive lists with touch support – in this case used to build the swimlanes,
- Bootstrap 3 is used together with custom CSS to give the application a well-tested base design,
- Node.js is a server runtime for JavaScript which runs the web server,
- Express is the web server which runs on Node.js,
- Mongoose is the Object-Document Mapper (ODM) that bridges Node.js and the MongoDB database (this project could just as easily have used relations, but documents are generally faster when working with JavaScript on the server),
- Body-Parser is used to convert JSON-formatted plain text to objects and back on the server, and finally
- MongoDB is the document database used to store issues

The front-end and back-end projects communicate using the Hypertext Transfer Protocol (HTTP) following the architectural style REpresentational State Transfer (REST). A sample implementation of creating an issue is implemented as follows:

Client-side service: ***nuissues/angular/js/issues/issues.service.js***

```
service.create = function(issue) {
    return $http.post('/api/issues', issue);
};
```

Server-side controller exposing a REST endpoint: at HTTP POST ¡host¿/api/issues***nuissues/node/controllers/issues.js***

```
router.post('/', function(req, res) {
        var issue = new Issue(req.body);
        issue.status = 'todo';
        issue.save(function() {
                return res.status(201).json(issue);
        });
});
```

Using the `ng-sortable`[7] module for AngularJS, the code for generating swimlanes in its entirety looks as follows where `validStatuses`, and `issues`, `swimlaneSortOptions` are defined by the page's AngularJS controller:

```
<span id="board" ng-repeat="status in validStatuses">
    <div class="jumbotron swimlane">
        <h2>{{status}}</h2>
        <p ng-if="issues[status].length === 0">No issues!</p>
        <div as-sortable="swimlaneSortOptions" ng-model="issues[status]" class="pre-scrolla
            <div class="issue-container" as-sortable-item ng-repeat="issue in issues[status
                <div as-sortable-item-handle ng-class="{issue: true, deleted: issue.deleted
                    <input class="title no-border" ng-model="issue.title" ng-change="issue
                    <span class="controls-delete">
                        <a ng-if="!issue.deleted" class="delete" href="javascript:void(0)"
                        <a ng-if="issue.deleted" class="restore" href="javascript:void(0)"
                    </span>
                    <div class="created static">{{issue.created | date: 'short'}}</div>
                    <!-- <a ng-if="issue.changed" href="javascript:void(0)" class="control-
                    <a href="javascript:void(0)" class="control-update" ng-click="updateIss
                </div>
            </div>
        </div>
        <div id="bottom-fade"></div>
    </div>
</span>
```

This code also shows the HTML implementation of the fade-out effect at the bottom of each swimlane, which is simply a `div` element styled with CSS as follows, assuming the file `bottom-fade.png` exists in the same directory as the CSS file (technique inspired by https://css-tricks.com/examples/FadeOutBottom/):

```
#bottom-fade {
        width: 600px;
        height: 200px;
```

[7] http://ngmodules.org/modules/ng-sortable

```
        z-index: 99;
        position: fixed;
        bottom: -20px;
        background: url("bottom-fade.png") bottom center no-repeat;
}
```

The live application is hosted on Heroku[8], which is a free host for Node.js (and other) projects.

2.7 Known bugs in the current implementation

There is at present time one known bug in the prototype implementation:

- It is nearly impossible to grab the bottommost issue that lies beneath the "fade out" overlay over the swimlane in a full swimlane

3 Evaluation of the system in general

The previous section presented the system that is NUIssues, relevant models, and planning behind the system. This section evaluates the prototype by looking at Wigdor and Wixon (2011, p. 34)'s guidelines for 2D spatial NUI design on how to support using 2D planar space, and some valuable outcomes and actionable events.

3.1 NUI guidelines

The only mandatory guideline for spatial 2D NUI design in Wigdor and Wixon (2011, p. 34) is that the design must:

> "Create an environment that is optimized for touch in its layout, feedback, metaphors, and behaviors. Any item that responds to users' touch must be at least 15mm in size in all directions, and there must be at least 5 mm between minimally sized touch targets".

NUIssues is true to this: on the target device, all *cards* have more than 5 mm margin to the next card, and each card is (much) more than 15 mm in all directions. This is demonstrated in 3.

[8] https://heroku.com

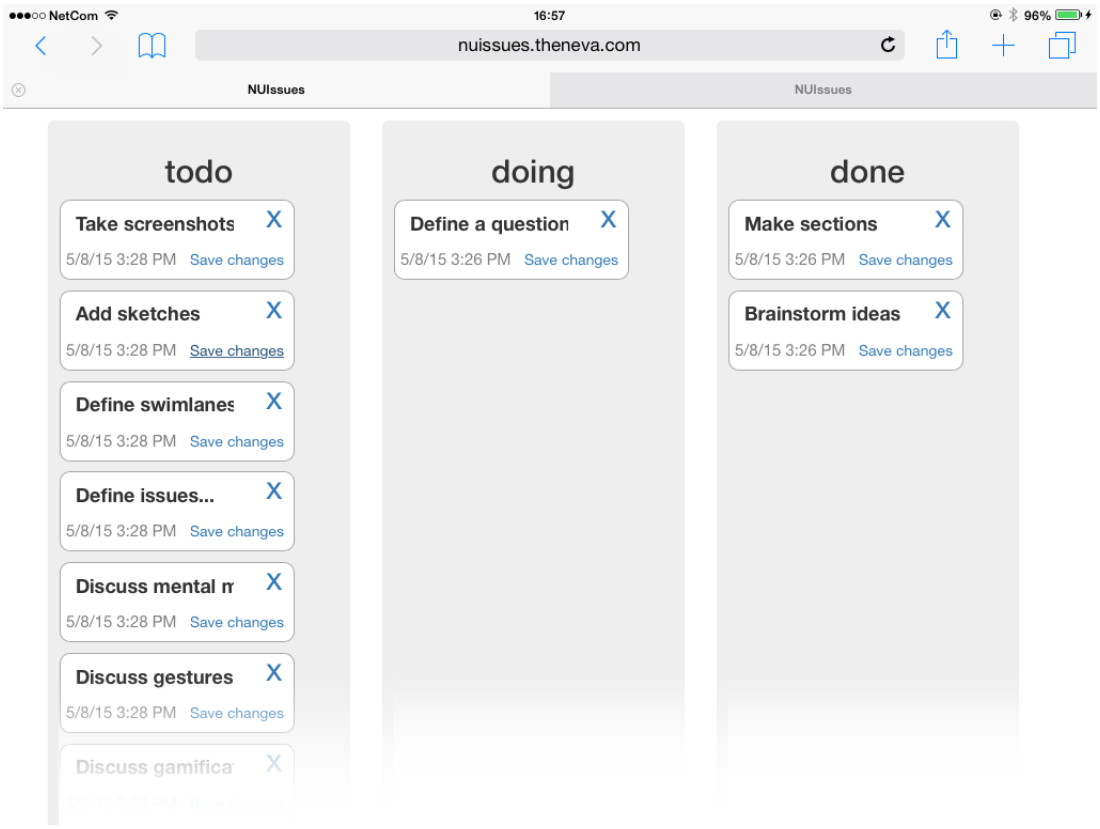

Figure 3: Standard view of NUIssues with issues in all swimlanes

A key concern here is scrolling within a swimlane, which is not hinted at in any way other than blank space to the right of the cards inside the swimlans. While more content is hinted at through a fading swimlane, only users familiar with touch interfaces with scrolling may possibly understand how to manipulate this list. This weakness is a substantial flaw in the design that should be revised in future iterations of the design.

The following points are *guidelines*, listed under "Should" in Wigdor and Wixon (2011, p. 34).

The design is consistent through the application, and the user is not allowed to customise the look of the app because the entire interface only consists of one single view.

The spatial relationships of the issues have been considered:

- In the Western world, we read from left to right, and thus the natural lifecycle of a card moves from left (*todo*) to right, ultimately reaching *done*.
- While not explicitly expressed in the design (so that the user is left to interpret the significance of the positioning), one may assume that the user's model of the system identifies issues higher up (with a lower index) as more urgent or important. After all, the grid layout imposes a hierachical style on the application. This interpretation is, however, intentionally left to the user.

NUIssues is not designed to be a multi-user system, although changs one user makes to the board affect every user of the same instance. Multitouch support would be interesting, but counter-intuitive in an application specifically designed for a personal tablet computer.

The main (and only) view has only the minimally required controls at any given time, and focuses on the *content* of the application – as does the user.

Finally, direct manipulation has been actively considered: manipulation always happens inline (only the title in the prototype), with no external prompts except the virtual keyboard on the tablet unless a physical keyboard is connected – which cannot be avoided.

Each card can be deleted individually, and accidental occlusion is prevented by highlighting the entire card if it is about to be deleted. One thing to note is that *only* using colour is not necessarily enough to indicate a status.

3.2 Potential sources of errors

There are several potential sources of error in a touch application. One of the most important, *fat fingers* as defined by Wigdor and Wixon (2011, p. 73) have been avoided to an extent by spacing and sizing elements in compliance with guidelines for 2D NUI design.

Other error sources include multiple capture states, accidental activation (like brushing the screen with an arm), physical manipulation constraints, stolen capture, and tabletop debris, which are all very difficult to counter with no real control over the actual use or environment for using the application.

3.3 Other notes and models

All objects on screen (except *swimlanes*) are interactive (editable, draggable, or sortable) through touch. Landing a finger on a card allows the user to drag the card to another position by sliding onto the closest new position and release the card to the new position by lifting the finger off the card. Landing a finger specifically on an editable text area on the card (the title) and immediately lifting the finger off, then editing the text with help of a keyboard. In this case a *gesture* (Wigdor & Wixon, 2011, p. 157) as an exit sequence would be helpful, but this is not implemented in the prototype.

While the application does not support any gestures, looking at Piaget's concept of the INRC group (Wigdor & Wixon, 2011, p. 137). This group defines four properties:

- Identity, stating that performing equivalent actions (like moving an issue to another swimlane) should reliably do an equivalent action;
- Negation, stating that it should be possible to return to the object's previous state after starting an action by performing the opposite action. For example, when dragging a card out of its place,

backtracing to where the card was picked up from and releasing the card will invariably return it to its origin.

- Reciprocal, stating that there are actions that return some aspect of the system to its original state. This is not implemented in NUIssues, but could take the form of an undo button.
- Commutative, stating that changing the order of independent operations does not affect the end result. For example, changing the title of an issue and then moving its card to a different swimlane ultimately results in the same system state as first moving the card and then changing the title of the issue.

3.4 Meaningful graphical states and actionable events

The entire application only has one single screen, but this view provides information in several different ways. Each state, or "view", will be provided in a screenshot, followed by a brief summary of the meaning of the state.

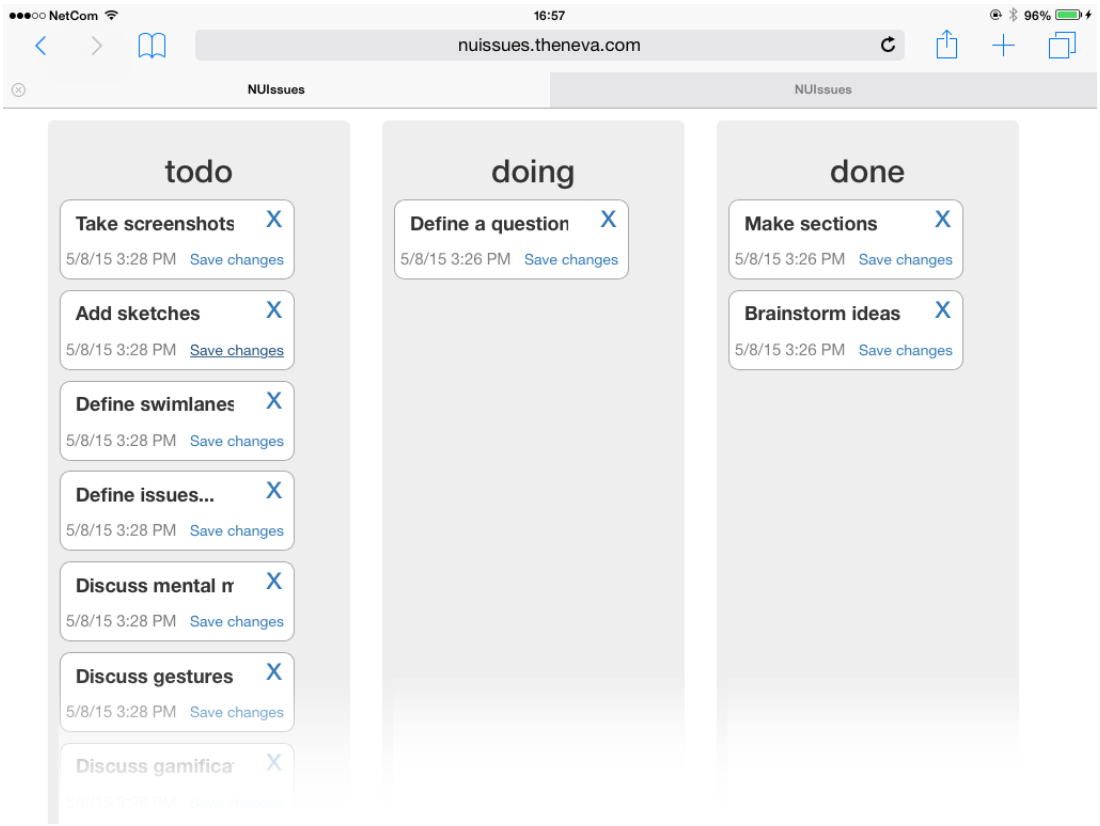

Figure 4: Default view of NUIssues with issues in all swimlanes

Figure 4 shows the default expected set of views. This includes *swimlanes* and *individual cards representing issues*.

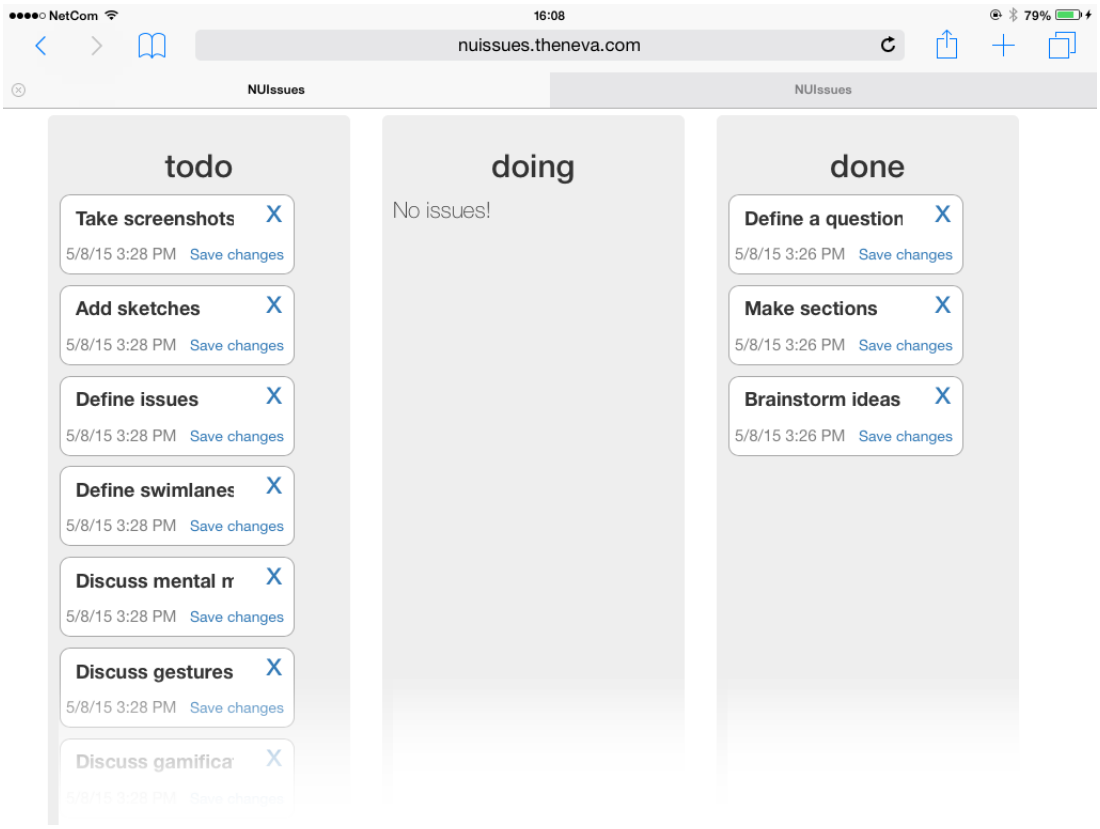

Figure 5: No issues in the *doing* swimlane

Figure 5 shows the feedback in response to a swimlane being empty. This hinders confusion with the issues actually being *loaded*, which hinders *apparent* system response times from being an error source by differentiating the two states.

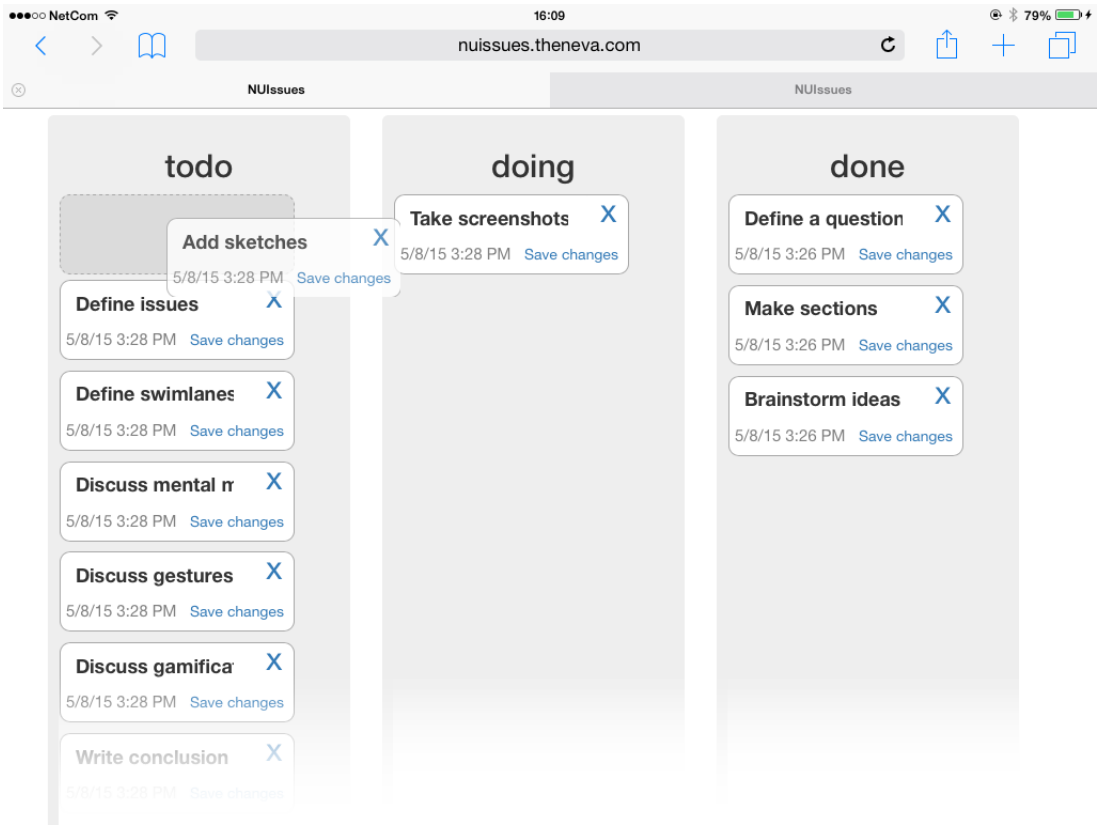

Figure 6: Card being moved

Figure 6 shows a card being moved, together with a hint indicating where the issue will land if released. The hinting always finds the closest position and indicates this position. Again, an escape sequence would be very useful here – but Piaget's property of *negation* is met.

- Issue that is marked for deletion (red: **occlusion**)
- Issue that is being edited (title)
- Hint to more content

Card deleted (realtime updates of the board to see what is going on?), but mostly "the time is X and we have Y tasks left to complete", or "we have way too many tasks in progress, please fix" (this falls under valuable outcome)

Potential: see who is assigned to the issue

Potential: see how long tasks are estimated to take, and prioritise based on that.

Potential for different input:

- Not using a mouse (but there's nothing in the way of using the app in a browser)
- Not using a stylus (although recognising a stylus and allowing in-place drawing of "attachments" for issues would be interesting to look into)

3.5 Valueable outcomes

3.5.1 Short-term valuable outcome

Immediately see and update the state of the project

Extremely simple interaction optimised for doing when travelling, can be used for walking, immediate overview of the entire project

3.5.2 Long-term valuable outcome

Be able to keep track of projects over time (but not across projects), be able to keep a continuous flow of issues

Focus on touch, more tactile approach, bring issues closer by letting people actually interact with them (tangible user interface?) physical form to digital information (which is really the basis of NUI in the first place, no? but not really, as there are no real objects; only digital visual representations of the actual work)

4 Conclusion and future work

NUIssues is a single-user system, and it would be interesting to explore multi-user issue trackers where the system for example presents several projects at the same time, and allows different teams to visually plan progress and coordinate work.

The *cards* in NUIssues are conventional, rectangular cards. An interesting future project can look into generation of iceberg targets (Wigdor & Wixon, 2011, p. 91).

5 Practicalities

Link to live instance of NUIssues: http://nuissues.theneva.com (hosted on Heroku).

Link to NUIssues source code: https://github.com/theneva/nuissues (hosted on GitHub).

References

Ishii, H. & Ullmer, B. (1997). Tangible bits: towards seamless interfaces between people, bits and atoms. *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, 411–415.

MacKenzie, I. S. (2003). Motor behaviour models for human-computer interaction in j. m. carroll (ed.), hci models, theories, and frameworks: toward a multidisciplinary science.

Norman, D. A. (2010). Natural user interfaces are not natural. *Interactions*, *17*, 6–10.

Wigdor, D. & Wixon, D. (2011). *Brave nui world: designing natural user interfaces for touch and gesture*. USA: Morgan Kaufmann Publishers.

Wilson, J. R. & Rutherford, A. (1989). Mental models: theory and application in human factors. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *31*, 617–634.