# Handling Design-Level Requirements across Distributed Teams:

## Developing a New Feature for 12 Danish Mobile Banking Apps

Lars Bruun, Mikkel Bovbjerg Hansen
Bankdata
Fredericia, Denmark
lars.bruun@bankdata.dk, mbo@bankdata.dk

Jørgen Bøndergaard Iversen, Jens Bæk Jørgensen, Bjarne Knudsen
Mjølner Informatics A/S
Aarhus, Denmark
jbi@mjolner.dk, jbj@mjolner.dk, bkn@mjolner.dk

***Abstract*—Bankdata and Mjølner have cooperated in the development of a new feature for 12 Danish mobile banking apps. Bankdata is the main system provider and Mjølner is subcontractor. Different teams from Bankdata have collected requirements, developed the necessary backend and middleware software, and designed the user interface. One team from Mjølner has implemented the app feature. The cooperation between the teams was centered around design-level requirements. Our contribution is to describe and discuss a number of lessons learned regarding requirements representations, requirements tools, and cooperation process; we have faced challenges, which were amplified by our distributed teams set-up. We also briefly describe a number of initiatives we have launched recently to alleviate the problems and improve the handling of design-level requirements in our future cooperation.**

***Index Terms*—Pragmatic requirements engineering, process efficiency, "good-enough" requirements, agile and lean approaches.**

### I. Introduction

Bankdata provides complete financial IT solutions for the banking sector. Bankdata is owned by a number of Danish banks and has approximately 650 employees. Services include tools for account managers, automated banking and home banking solutions for desktop and mobile devices. Mjølner develops custom-made software solutions for Danish and international customers. Mjølner has expertise in development of a broad range of system types, among them mobile solutions. Mjølner has around 80 employees.

Bankdata uses Mjølner as a subcontractor for app development. Since early 2013, our companies have cooperated on making a new feature for mobile banking apps for 12 banks. The feature is called Swipp. It allows money transfer between two persons, using a smartphone and only based on knowledge of the recipient's phone number. The first version was released to the Danish market in the summer of 2013. Since then, two subsequent versions have been developed and released. In March 2014, Swipp has around 350,000 users.

In this paper, we describe our cooperation in the Swipp project, which has been organized in distributed teams. All authors have participated in the project, and together we represent the roles of user experience designers, business analysts, app developers, and project managers. From Bankdata, approximately 15 people spanning three teams have been involved. The project also included backend developers and software testers, in addition to graphical designers from another subcontractor. Mjølner had one single team consisting of four to six app developers and a project manager. Approximately 12,000 project hours have been used producing around 25 different new app screens.

Lauesen [4] classifies requirements in goal-level, domain-level, product-level, and design-level requirements. This project has primarily been concerned with product- and design-level requirements, and the main focus in this paper is on design-level requirements, i.e., the detailed specification of the user interface (unless anything else is specified, the term requirements refer to design-level requirements).

Most of our discussions in this paper involve the handling of design-level requirements internally between the distributed teams that together constitute the development organization (we refer to this as internal communication). To provide some background information, we also give an introduction to the general requirements process between the development organization and other stakeholders (external communication).

External communication is extensively described in the requirements engineering literature. However, we are not aware of many authors who have discussed the internal communication issue. Efficient internal communication is crucial on the long and complex path from initial ideas about requirements to realization

in the form of adequate solutions that are well aligned with the needs of the users.

The structure of this paper is: In Section II, we present the mobile banking apps and their context in general; this includes a description of the Swipp feature in more detail. Section III describes the general requirements process and the requirements artifacts and tools we have been using. In this way, Section II and Section III constitute background material aimed at giving the reader an overall understanding of the system we are considering and the process our work is part of. In the central Section IV, we list and discuss a number of lessons learned, including the improvement initiatives we have launched. The conclusions are drawn in Section V, which also includes a brief discussion of related work.

Even though our process, as we will discuss in more detail, had room for improvement, the Swipp feature was delivered on time, with few errors, and has been very well received and rated by the users.

## II. The Apps and their Context

The mobile banking apps are available on the iOS and Android platforms. They have been developed in parallel and have a history of four years. The apps are for 12 different banks; one of these is among the largest banks in Denmark.

There is only one codebase per mobile platform, iOS and Android, because it has been decided that all apps should have the same functionality. The majority of the apps vary in appearance, e.g., details like theme color, bank logo and other bank identity properties, which are easily parameterized. The app for the largest of the banks has been further individualized with a different main menu user interface, different graphical design, and more app preference pages with options for user interface customization.

### *A. Functionality*

Frequently used features in the apps are viewing account balances, account entries, transferring money, and paying bills. Moreover, the apps have utility functions like currency lists, a currency calculator, and a function to block your credit card. In total, each app has approximately 100 different screens (about 25 of these are related to the Swipp feature).

A large and complex set of functionalities in the apps is the investment features. You can use this to track the stock exchange market, overview your securities and sell and buy securities. Exchange rates are visualized with interactive graphs.

The latest feature, Swipp, is the largest in the apps in terms of lines of source code and the number of app screens. The feature enables users to transfer money just by using the recipient's phone number. Account numbers are not needed, which is convenient as one person typically does not know another person's account number, and they tend to be hard to remember. When a payment has been registered, recipients are instantly notified about the transaction. Fig. 1. shows a Swipp payment screen from one of the apps.

In order to use the Swipp service, you have to sign up first. To sign up, the user completes a wizard and chooses which phone number to use, selects an account to assign to the agreement, and enters some additional details. Multiple agreements can be created and managed.

To make a Swipp payment, you enter a phone number, select a previous recipient, or select a recipient from your phone contacts, which are listed in the apps. Then you enter the amount to transfer.

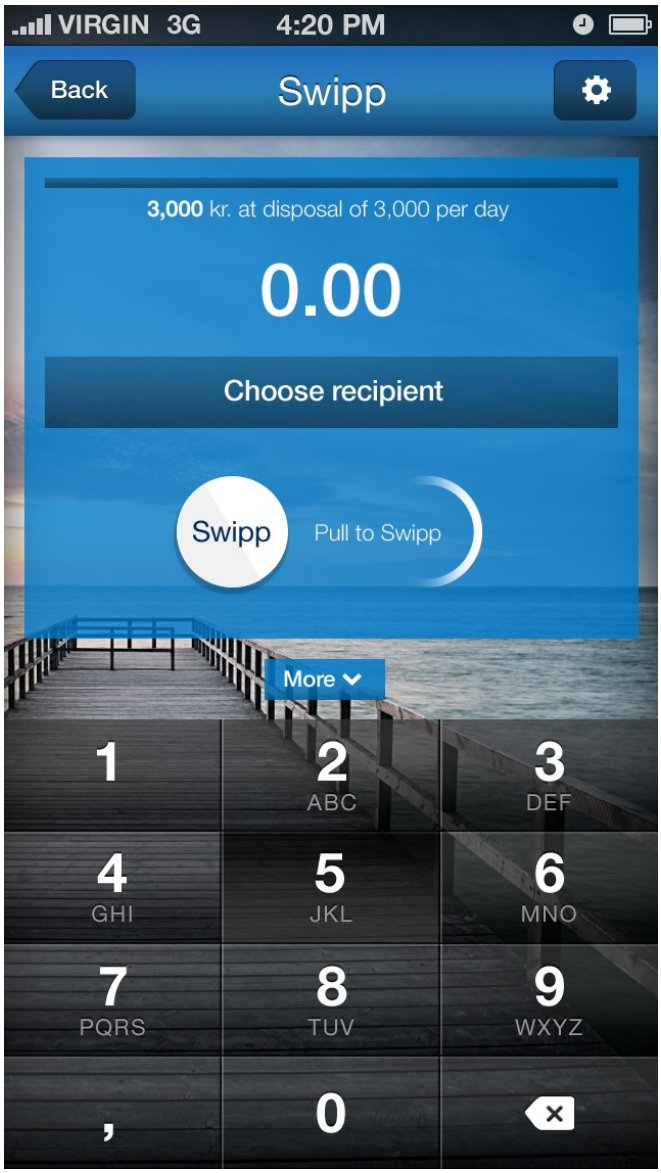


Fig. 1. Swipp payment screen from one of the apps (collapsed state).

### *B. Integration of the Apps with the Backend Bank Systems*

All the apps integrate with the bank systems through a shared middleware interface. The middleware component is implemented in Java and has a REST interface.

The middleware interface is a borderline for division of work between Bankdata and Mjølner. Bankdata is responsible for development of the middleware including adding new service

methods and modifying existing ones to be able to handle the data requirements of the app. A team of developers at Bankdata are assigned to work with this component. Mjølner handles the development of all apps.

The middleware is a thin layer, which is acting as a facade to the backend component, which holds all the core business logic and data. The backend is also servicing other components like the home banking solutions and internal bank applications.

The backend is a COBOL component. The people working with this component are in a different team than the ones working with the middleware component. In many cases, the backend component already supported the functionality required by the mobile apps. Sometimes, however, it was necessary to change the backend component, so it could be used by the middleware to deliver an efficient service to the mobile apps.

The overall composition of the system is illustrated in Fig. 2.

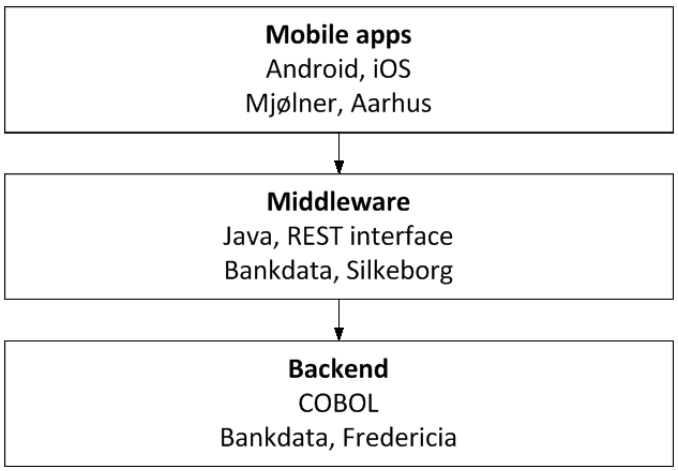


Fig. 2. Component diagram. The figure also indicates the different geographical locations with Mjølner in Aarhus and Bankdata in two cities, Silkeborg and Fredericia.

## III. General Requirements Process, Artifacts, and Tools

A wide range of stakeholders were involved in order to specify requirements. Goal- and domain-level requirements were established by a joint venture consisting of a vast majority of the Danish banks. An example of a goal-level requirement could be that the banks should gain market shares on the very competitive market for making mobile payments; an example of a domain-level requirement could be that it should be possible to transfer money to another person just using a mobile phone.

For most parts those requirements could not subsequently be influenced by the individual banks and their IT service providers such as Bankdata. The majority of the requirements were passed on to Bankdata at the beginning of the project and remained relatively stable throughout the course of the project. All requirements from the joint venture were documented in textual form supplemented by high-level process diagrams.

At Bankdata, the project was divided into three separate project teams with project members from different departments situated at two different geographical locations. Requirements were analyzed and specified primarily in cooperation between Bankdata, the subcontractors, and various representatives from Bankdata's customer banks. Product-level requirements were typically identified at workshops with the Bankdata project team members and the banks.

After that, the requirements were incrementally detailed in cooperation between business representatives and project team members. In this way, the design-level requirements were specified. Mjølner was normally not involved at this point in the process. The requirements delivered to Mjølner were detailed and documented using:

- Use cases (prose and diagrams)
- Interaction design (wireframes)
- Graphical designs (image files)
- Middleware interface descriptions (prose)

In order to avoid multiple use case descriptions because of what was believed to be minor user experience deviations between different devices, it was attempted to keep use case descriptions generic and device independent, i.e., the same use case descriptions were used for development of both the iOS and the Android apps.

The project use case descriptions were intended to have a detail level similar to that of Cockburn's [1] fish-level description. The use cases contained, e.g., verbal and graphical descriptions of the actors involved, the main flow, and all alternative and exception flows. Use case descriptions were also supplemented with lists of functional requirements (business rules), as illustrated in Fig. 3.

| ID | Description |
|---|---|
| FR-03 | The following input fields are mandatory:<br>*Amount<br>*Mobile phone number |

| ID | Description |
|---|---|
| FR-27 | Recipient (mobile phone number) must be active in database |

Fig. 3. Extract from list of functional requirements from the 'Execute Swipp payment' use case.

An example of a flow description in the form of a UML activity diagram is shown in Fig. 4.

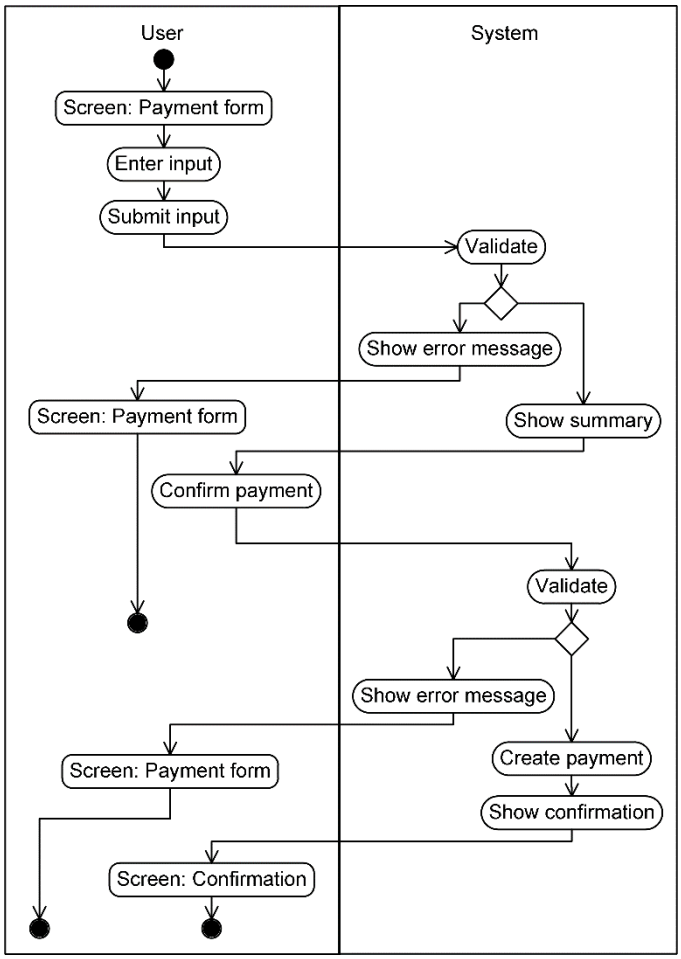


Fig. 4. Simplified version of activity diagram for the 'Execute Swipp payment' use case.

Interaction design [8] was documented with wireframes, which were combined into a navigation diagram including main and alternative navigation flows. An extract of the navigation diagram is shown in Fig. 5.

Wireframes and navigation diagrams were only used to document navigation flows, content, and the structural aspects of the interaction design. The look and feel was specified in the graphical design files; an example is shown in Fig. 6.

The exchange of the requirement artifacts and the parties involved are illustrated in Fig. 7. As can be seen from the figure, the following teams were involved in the requirements handling:

- Business analysis and user experience design team at Bankdata
- Middleware development team at Bankdata
- Backend development team at Bankdata
- App development team at Mjølner
- Graphical design team at another subcontractor

This distributed teams set-up was a major contributor to a number of the challenges we experienced regarding handling the design-level requirements, as we will discuss in more detail in the following section.

Fig. 7 emphasizes the difference between external communication and internal communication, cf. the discussion in the introduction. The external communication is represented by the topmost arrow. The internal communication is represented by all other arrows.

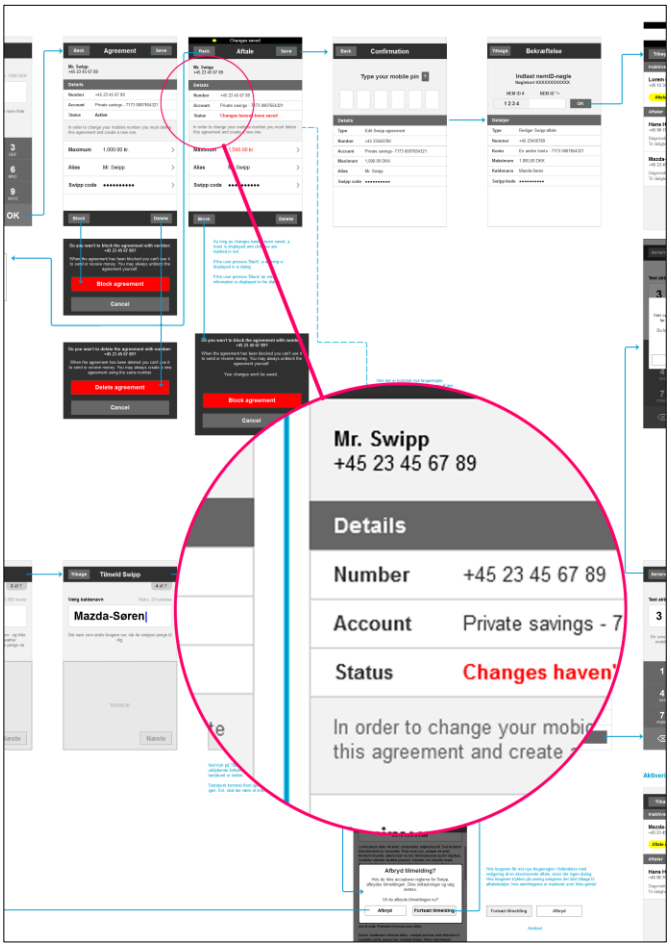


Fig. 5. Section of wireframe navigation diagram.

## IV. Lessons Learned

After having introduced the apps and the general requirements process, artifacts, and tools, we now list and discuss a number of lessons learned from the development of the Swipp feature.

### *A. Too Many Requirements Representations*

The multitude of document types used for specifying design-level requirements and the distribution of those documents across several different repositories proved to be a major challenge throughout the project.

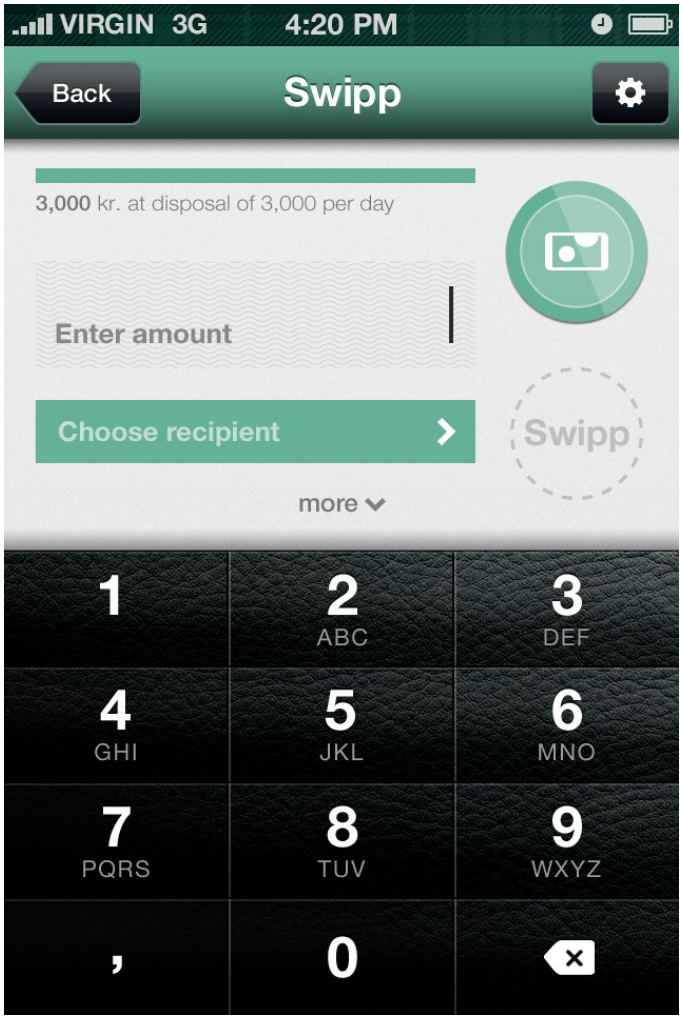


Fig. 6. Graphical design for a Swipp payment screen from one of the apps – the design varies from the design depicted in Fig. 1.

In many cases, we did not manage to avoid redundancy between the different documents. The position of an input field, e.g., would be described in four different documents: 1) a wireframe - and so would any associated label and placeholder text. In order to specify related business rules and any conditions regarding data formats and validation, the input field was referenced in 2) the use case descriptions. The visual look of the input field was illustrated in 3) the graphical design files - once again including possible labels and placeholder texts. Finally all requirements, except for those related to graphical design, would be integrated in 4) test case descriptions. As a result of this, several different documents in different repositories had to be updated because of something as trivial as the change of a placeholder text in an input field.

These conditions were further complicated by the fact that certain representations were generated from specialist tools to more viewer-friendly formats.

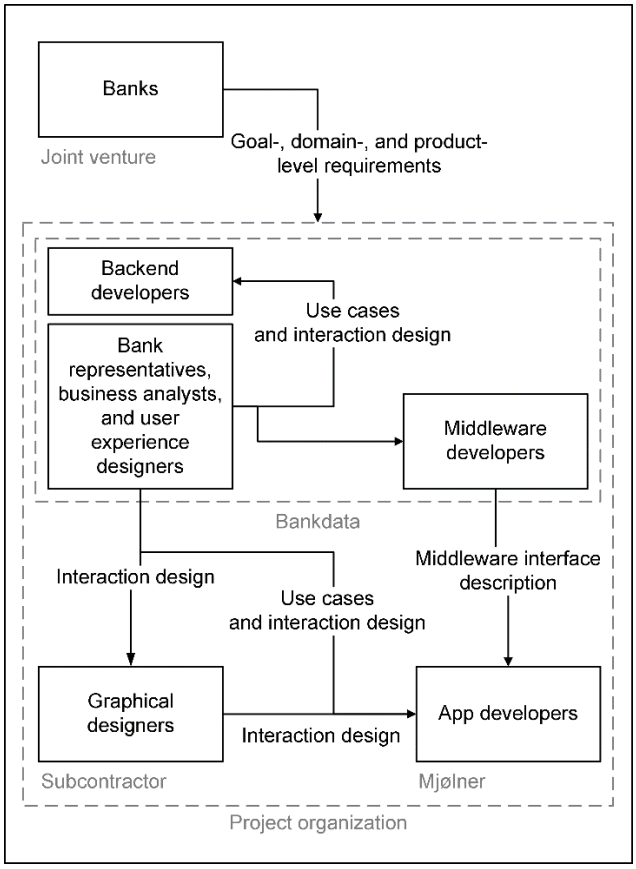


Fig. 7. Exchange of requirements artifacts and the parties involved.

Wireframes and navigation diagrams were distributed as html-prototypes, Word document specifications, or image files generated from Axure RP (a tool supporting generation of html prototypes, Word specifications, and image file wireframes). Graphical design files were distributed as image files generated from Photoshop.

This redundancy introduced a number of sources of errors and constituted a built-in inertia that hampered the process of updating and synchronizing requirements representations.

Some inconsistencies between the different representations were deliberately accepted. In order to minimize resources spent on graphical design, e.g., it was decided that graphical design files were only updated, if changes in requirements strictly involved the look and feel of a user interface element, i.e., shape, size, and tint, but not changes of phrasing, values, etc. Allowing such inconsistencies in the available documentation may seem rational and reasonable but still, however, proved to be a problem. Even when changes were made in the wireframe documents only, the set-up was not favorable. As soon as graphical design files were updated to newer versions than their wireframe counterparts, misunderstandings easily emerged. When a design file was updated (e.g., because of a new icon design) it simply proved counterintuitive to disregard the now deprecated text labels in the document, because it had a newer version date than the corresponding wireframes.

We have not been able to omit any of the different documentation types entirely. In order to eliminate the alignment

problems, we are working on reducing information redundancy. At a closer inspection, we have noticed that our use case descriptions may be too detailed. In certain aspects, the detail level is closer to clam-level than the intended fish-level, cf. Cockburn [1]. In future projects, that level of detailing will be avoided in the use case descriptions and delegated to the documentation of the interaction design in the form of wireframes.

In order to reduce the redundancy between wireframes and graphical design we have chosen that the final graphical design is primarily documented as component sheets and style sheets (depicting no real data). This way redundancy is removed and the different purposes of wireframes and graphical design are further emphasized.

### *B. Cooperation Process and Tool Support Were Insufficient*

As described in Section III, the requirements package delivered to Mjølner contained the following artifacts: use cases, wireframes, graphical design, and a middleware interface description. The artifacts in question were created by different project members at Bankdata, which in itself poses a number of risks related to alignment of documentation.

Alignment problems, however, typically occurred after Mjølner made their initial reviews. During Mjølner's review of wireframes and use cases, mistakes and improvements were identified. Hereafter the wireframes and use cases often would get out of alignment due to insufficient communication between the different project members responsible for the relevant artifacts at Bankdata. Sometimes, this led to confusion among the Mjølner development team and the Bankdata team.

The regular communication set-up in the individual organizations could not easily be integrated with the distributed teams. Therefore, other communication tools had to be used, and since insufficient attention had been given to the communication set-up when the project was established, the needs were taken care of as they emerged. Because of that, we ended up using four different communication channels: Email, an FTP-server, a Sharepoint site, and a task management system. Mail communication in itself of course is not befitting for findability and a set-up with so many communication channels creates barriers that does not favor alignment of information.

When the implementation of the apps began and the testing of the first versions was initiated, the project needed to handle a number of requirement clarifications on a daily basis. Because the project teams were situated at different locations, the communication was primarily handled via a shared task-management system and by email and phone.

The various project teams (including the subcontractors) had several points of contact and there were no sufficiently well defined procedures for communication and documentation maintenance. When changes or further detailing of requirements were made, the original specifications were often not updated. Either because it seemed redundant or because it was unclear who was responsible for updating the documentation. Especially software testers experienced problems with keeping the test cases up to date because in many cases they were not involved in the decision making relevant for design-level requirements and relied entirely on specifications and briefings from the other project members.

As part of Bankdata's test effort, they reported several bugs in a task management system shared with Mjølner (Jira). Some of the bugs lead to further clarification of the requirements related to the buggy feature, and this was communicated in comments to bug reports. The problems experienced with email clarification were also present when the clarification was initiated by a bug report and the discussion took place in bug report comments.

All in all, the documentation of the requirements was spread across a number of documents and repositories and distributed via several communication channels. This resulted in a number of problems: Finding the latest information about a requirement could involve traversing documents, images, emails, bug reports, and talking to colleagues. Contradicting descriptions of the same requirement could exist in the documentation, because different people were responsible for updating the documentation and because documentation was distributed across different repositories.

Much attention has been given to the future communication process that should explicitly support distributed teams. It has been decided to use a task management system as a hub for all project communication. VersionOne has been chosen since it is already implemented at Bankdata as the corporate project management system. Although documentation still resides in multiple repositories, the task management system will be the focal point for all project members. Documentation is referenced from the relevant feature groups, use cases, and tasks in the task management system, and relevant project members are automatically notified when task conversations are updated. Project members have also been asked to document decisions made at meetings or by phone in the task management conversations. Communication by email will be kept at a minimum, and it has been decided to use a shared mailbox to communicate with external stakeholders that do not have access to the task management system.

In addition to improved tool support, Bankdata and Mjølner have planned to work at the same location once a week. This is expected to enhance the communication process.

### *C. Software Developers Were Not Consulted Sufficiently Early*

During the project, there were several incidents where implementing a component became more time consuming than expected. Often this was because the user experience designers were not sufficiently aware of the specific technical challenges in implementing particular components. We give two examples to illustrate this.

The first example involves the development of a custom component. To match the design of the app, it was decided to use a custom-made keypad instead of the operating system's built-in keypad. For outsiders it may seem like a trivial task to build a component with buttons for the digits zero to nine and a few additional characters. But for the app developers, the task was far more complex. Intricate behaviors like opening and

closing the keypad, adjusting to different screen sizes, scrolling the input field into focus, etc., often go unnoticed. All these behaviors are handled nicely by the built-in keypad, but had to be developed anew for the custom keypad. The result was that it was far more time consuming to develop this part of the app than first estimated.

Another example relates to a graphical redesign of the Swipp payment screen. At a first glance, the redesign introduced what seemed to be minor visual differences between the individual apps. But when it came to implementation, those differences added a significant overhead to the development, because many subparts of the graphical layout for the different apps had different screen positions. The developers ended up maintaining two different versions of the screen, instead of just changing style and image files. The result was more time spent on the redesign than anticipated.

In both examples, there had been a better basis for making a trade-off between app quality and effort required for implementation, if the app developers had been involved earlier, i.e., if there had been a better communication between the distributed teams. When the developers made their observations, Mjølner assumed that it was too close to the deadline to suggest changes to Bankdata, because they would have to involve the banks for approval, which would require too long calendar time. The cost of involving more people in the earlier stages of the project is likely to be less than the extra time spent in the implementation stage. Furthermore, we believe that involving the developers at an early stage could have led to more innovative solutions to certain challenges, because of the extra knowledge brought to the discussion via the developer perspective.

We have changed our cooperation process, and in future development projects the app developers will be involved as soon as business analysts and user experience designers are ready to present the first sketches of desired user experiences. The weekly workdays on the same location will hopefully ensure regular technical assessment of the desired user experience designs by app developers.

Middleware interface specifications is another subject, where app developers have been involved too late. Sometimes the middleware interface description was made after the development of the actual middleware interface. The middleware interface description was not always carefully reviewed before handed over to Mjølner, in order to verify that the middleware interface description were consistent with the wireframes and use cases.

One example of this was a set of functions involving security signing of new Swipp agreements created on the phone. In the first version of the interface, it was possible to sign an agreement, but the middleware did not support the flow through the app that the wireframes described. After some discussions between the middleware developers and app developers, a suitable compromise was reached.

One of the reasons for the lack of reviews was a result of the fact that the middleware development was not a fully integrated part of the Swipp project. Because of other assignments, the middleware developers were not allocated to the project full time, which meant that the middleware development tasks were not always well coordinated with the rest of the project activities.

Because of this, the middleware development has now been fully integrated in Bankdata's mobile banking project organization.

### *D. Platform-Dependent Design-Level Requirements Were Not Considered Sufficiently Early*

The iOS platform has always been the starting point when interaction design for the apps has been made. All the mock-ups, concept design and pixel perfect screens have been carefully tailored with great respect to iOS guidelines and a lot of resources have been used to make high quality designs for Apple devices.

When the iOS design was completed, the Android developers gave inputs to changes to make the Android apps less iOS-like and more in line with Android design principles. One example of this is a simple user interface control like an iOS on/off switch, which on Android should look and function more like a standard toggle button. Another example is the use of back buttons and top menu buttons in iOS, which on Android should be removed because of the hardware buttons.

The practice of designing for iOS first and using the same design for Android is very common, and is reflected in the design of a lot of the apps in the market. There are a number of reasons to do this.

Statistics for various apps, that we have been involved with, have shown much higher iOS download and traffic numbers for iOS users compared to Android users. Therefore iOS devices are often targeted first and the interaction design is then subsequently reused for Android devices in order to save resources or because the iOS design becomes an implicit point of reference for the interaction designers.

Another reason why iOS design often precedes Android design is that iOS apps have more strict submission criteria. Previously there were also considerably more design guidelines for iOS than Android, but even though this has changed, still almost anything will be allowed on the Android app market, so there is less need for tailored design on that platform.

The need for design customizations for Android apps has prior to this project been fairly manageable and adjustments were handled on the fly. In the development of the Swipp feature, however, we encountered more fundamental disparities between the interaction design suited for Android and iOS. Some of the challenges revolved around the fact that the Android ecosystem is much more fragmented than iOS in terms of display dimensions. In certain contexts, this becomes particularly important to take into account when designing.

For the Swipp feature we used new layout concepts different from the existing concepts used in the apps. An example of this is the Swipp sign up wizard, where the content of each page had been super optimized to the very limited range (currently two) of iOS display sizes. In order to draw attention to content that was hidden behind the keypad on small iOS displays, a button to hide the keyboard and make hidden content visible was introduced. This design, unfortunately, is based on a fixed layout

concept, which from an implementation perspective is not feasible when it has to be applied to a lot of different screen sizes. The latter is characteristic for the excessive amount of different Android devices.

For each of those fixed screen designs it was necessary for the Android app developers and the user experience designers to investigate and find alternative solutions to make the design work on Android. The general solution has been to handle three different groups of display sizes differently. A drawback to this solution is a more complex code base, which will require more maintenance and testing. The advantage is that Android devices with the most general screen sizes have the same layout as iOS devices.

To avoid legacy issues, e.g., with iOS designs, which do not fit well on Android devices, a couple of new approaches have been planned. In the first place, app developers for each platform will be involved in the early stages of the design process and attend meetings, where early sketch designs are presented. A more timely developer assessment of preliminary designs can ensure that layout concepts that are inappropriate from an implementation point of view are rejected. Secondly, the interaction design will be tested on multiple display sizes before it is finalized. Furthermore, the user experience designers are investigating how best practices from responsive web design can be transferred to app design in order to produce more fluid layouts that will accommodate a broader range of display sizes and aspect ratios.

These changes in the process will ensure that design-level requirements for all relevant platforms are not overlooked and also provide a better foundation for estimates.

## V. Conclusions and Related Work

To sum up, the lessons we have discussed above are that 1) there were too many requirements representations, 2) that the cooperation process and tool support were insufficient, 3) that software developers were not consulted sufficiently early, and 4) that the platform-dependent design-level requirements were not considered sufficiently early.

We believe that the distribution of work across different teams has been a primary reason for the problems we have experienced. If, e.g., it had been possible to carry out the project with one single team with 6-8 persons, the problems would most likely have been less severe. In a dogmatic agile project, there are no teams distributed across different geographical locations – not even different rooms. This of course means that the needs for elaborate documentation - which may cause redundancy – and formalized communication procedures – with another balance between written and oral communication - are quite different from those of geographically dispersed development organizations such as ours. We have realized that we until recently have underestimated some of the challenges caused by the size of the project and by the distributed teams set-up.

Our project was planned to apply a number of key elements from agile development, e.g., iterations of a few weeks' length and frequent demonstrations for bank representatives. This would allow us to gain feedback that could be used to make prioritizations for the next iterations of the project. In this way, we have in fact dealt with a number of requirements issues that have emerged throughout the project. However, there have been stages in the project, where this way of working was not fully enforced, and where feedback or other cooperative measures therefore have come too late, as we have discussed.

In addition, we have not had one explicitly appointed product owner as a central point of contact regarding requirements issues, which might have contributed to an alleviation of some of our problems. We have decided to introduce a product owner now, but we realize that such centralization may cause new problems, like introducing communication bottlenecks. Whether the advantages balance well with the drawbacks are to be seen. This remark generalizes to all the improvement initiatives, we have launched; we do not have much evidence about their effect at the time of writing this paper.

As mentioned in the introduction, we are not aware of many authors, who have described the challenges involved in handling internal communication across distributed teams in the requirements engineering literature. We do know of a couple of papers though. Gross and Doerr [3] discuss the needs for different representations of requirements for different roles. Marczak and Damian [6] set a theme which is similar to [3], and have in previous work explicitly addressed how distributed development influences cooperation patterns [2].

Regarding the lesson about not getting the software developers involved sufficiently early, it is widely recognized that requirements and software architecture are interdependent subjects that should not be treated separately or in a given, sequential order [7]. This relationship is, e.g., discussed by Loft et al [5] in the context of a project involving Mjølner, where the importance of very close cooperation between requirements engineers and software architects from the beginning of a project is described. In comparison with the Swipp project of this paper, this may be easier to facilitate when requirements engineers and software architects work within the same company and location, such that interactions do not have to cross organizational boundaries and physical separation.

A few remarks about the RE14 theme of innovation. We believe that the Swipp feature – although a rather straightforward concept – is an innovative approach to making payments. Swipp in itself is not the result of an innovative requirements process, but we think that we must focus on being increasingly innovative and creative in the ongoing requirements and development process in order to cope with the tough competition in mobile banking and mobile payments.

We hope that the improvement initiatives we have launched will be beneficial – and that they are more generally applicable and can be used in other, similar projects. In particular, we believe that projects with distributed teams must give much attention to ensuring good and efficient communication processes and strive to reduce redundancy in requirements representations.

## ACKNOWLEDGMENT

We thank all our colleagues at Bankdata and Mjølner, who have contributed to the project or who have read and commented and suggested improvements to this paper.

## REFERENCES

[1] A. Cockburn, Writing Effective Use Cases, Addison Wesley, 2000

[2] D. Damian, S. Marczak, I. Kwan, "Collaboration patterns and the impact of distance on awareness in requirements-centred social networks", RE07, New Delhi, India, IEEE, 2007

[3] A. Gross, J. Doerr, "What you need is what you get: the vision of view-based requirements specification", RE12, Chicago, Illinois, IEEE, 2012

[4] S. Lauesen, Software Requirements - Styles and Techniques, Addison Wesley, 2004.

[5] M. S. Loft, S. S Nielsen, K. Nørskov, J. B. Jørgensen "Interplay between requirements, software architecture and hardware constraints in the development of a home control user interface", Twin Peaks workshop at RE12, Chicago, Illinois, IEEE, 2012

[6] S. Marczak, D. Damian "How interaction between roles shapes the communication structure in requirements-driven collaboration", RE11, Trento, Italy, IEEE, 2011

[7] B. Nuseibeh, "Weaving together requirements and architecture", Computer, pp. 117-117, Mar. 2001

[8] H. Sharp, Y. Rogers, J. Preece, Interaction Design, John Wiley & Sons, 2007