Naked Objects versus Traditional Mobile Platform Development: A Comparative Case Study

Heikki Keränen[1, 2] and Pekka Abrahamsson[1]
[1]*VTT Technical Research Centre of Finland*
PO Box 1100, FIN-90571 Oulu, Finland
[2]*Department of Information processing science*
FIN90014 University of Oulu, Finland
{heikki.keranen; pekka.abrahamsson}@vtt.fi

Abstract

It has been suggested that use of the Naked Objects pattern could contribute to business agility of applications and reduce the amount of the application code up to 75 percent in a desktop environment. Very little empirical evidence exists to support these claims, however. This paper reports results of a study where two agile software development projects created the same mobile application using two different technologies. The first project did the development using the traditional mobile platform, and the second project used the Naked Objects Framework. Both projects used the same agile software development process. The results show that Naked Objects produced 79 % less application code and 91 % less user interface code. Yet, the version of the Naked Objects Framework used was not found to be mature enough for proper implementation of all required functionality. It is concluded that, if further developed, Naked Objects can be a very potential technical development platform for business-oriented applications. Implications of the study are addressed.

1. Introduction

Naked Objects [6] is an architectural pattern which exposes core business objects to a user. The pattern implementation is supported by a toolkit called the Naked Objects Framework. Pawson and Matthews claim that the Naked Objects technology has many advantages over the traditional object oriented development principles. These include reducing the length of the development cycle and empowering the user as a problem solver. Increased business agility is achieved, since even radical modifications from the business point of view are implemented rapidly, with small changes, to the system [6]. Pawson and Wade [8] have suggested that the Naked Objects pattern fits well into the agile development principles early in the project when the user requirements are explored. The recent developments of the Naked Objects technology are improving the framework so that it can be successfully used to carry out the development of a full working release of the software.

Very few empirical analyses of the Naked Objects technology are, however, available. Pawson [7] conducted one comparative implementation for a desktop application, which showed a dramatic 75 % reduction of the application code size. Bettin [3] speculated that the use of Naked Objects may reduce manual coding effort up to 80-95 %. It is not very clear from the Bettin's study what the empirical setting was. It appears that he used theoretical devices (i.e. rather than empirical) to come up with his results. However, these results alone are not sufficient for a convincing empirical body of evidence but they serve as a good motivation to study the Naked Objects technology further.

Myers and Rosson [5] pointed out that nearly fifty percent of the code in applications is devoted to the user interface. One of the key ideas in the Naked Objects technology is that major portions of the user interface are automatically generated, which helps in saving significant amounts of code.

In this study, two agile projects developed the same mobile application, which enabled users to access the Helsinki Stock Exchange for trading and viewing stock markets in a Java enabled mobile phone. Both projects utilized the same agile software development method. The first project utilized a traditional mobile development environment and the second one utilized the Naked Objects Framework. The functionality requirements for the applications were the same. Thus, this provides a good basis for comparing the results

Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05)
0-7695-2431-1/05 $20.00 © 2005 IEEE

and finding evidence about the validity of Pawson's [7] and Bettin's [3] claims.

This paper presents a lot of empirical data, since so little empirical data on the Naked Objects -based development exists. Thus the data is of exploratory type and serves as a reference for other researchers and practitioners.

The paper is composed as follows. The next section describes the Naked Objects technology. This is followed by a description of the research design and the results of the study, after which the results are discussed and implications are drawn.

2. Naked Objects Framework

In this paper, traditional mobile software platform is consisting of the industry standard platform components. These components include Java Mobile Device Information Profile (MIDP) (mobile phone side) and, on the server side, an SQL database and Java 2 SDK, Java Server Pages, Java 2 Enterprise Edition.

When using these techniques, the software is typically divided into a user interface layer (client), a server layer and a database backend. Regardless of the selected components, the developers have to develop all these layers and implement a communication protocol between them. Usually, a business level modification to the application causes modifications to all of these layers. Often, the implementation of the user interface takes the most of the effort.

The Naked Objects Framework is a Java-based software framework which helps to utilize the Naked Objects architectural pattern. In the Naked Objects pattern, the core business objects encapsulate all business data and behavior. In the Naked Objects Framework, they implement the Naked Object Java interface, and the core business objects obey a set of simple coding conventions. The framework has an Object Viewing Mechanism (OVM) which autogenerates a user interface based on information contained in the business objects. The core interfaces implemented by the application and the Java reflection mechanism provide the information for the autogenerated user interface. [7].

Due to the abstract nature of Naked Objects, it is possible to create OVMs for different kind of devices. This study utilized an OVM for MIDP called the MIDP-OVM[1]. To complement the autogenerated UI, a set of specific views can be developed for selected business objects, if necessary.

1 Freely downloadable from http://opensource.tte.erve.vtt.fi/pdaovm/midp-ovm/

The Naked Objects Framework also contains a set of Object Stores, which provide automatic persistence for the business objects. This study used an XML Object Store, which persists the business objects into a set XML files.

3. Research Design

This section describes how the research design for the study is laid out. The research method is a comparative case study [9].

In both projects, the development was guided by an adapted version of the Extreme Programming approach called Mobile-D [1]. The adaptation has been performed taking into account the specific demands (variability, memory use, screen size, etc.) of the mobile development environment. The development proceeds in one to two-week iterations. Each iteration is divided into three parts: planning day, working day and release day. Thus, in a single iteration, there is one planning day and one release day and a multiple number of working-days.

3.1. Stock Exchange Application

The outline of the developed software is presented in Figure 1. Stock market information is parsed from the public web pages and is stored into the database several times a day. The user can access this data using a Java client in the mobile phone. In addition to the public market data, a user can enter information about his investments to the system and see their current value and daily changes. The user can also set alerts to share prices. The user can also trade in the system.

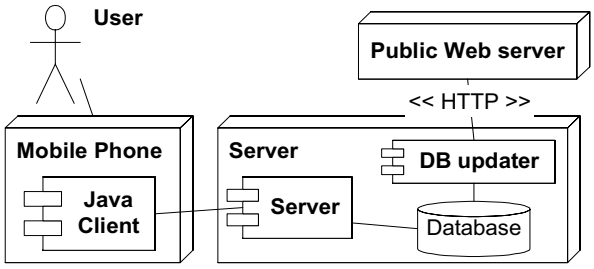

Figure 1. Deployment of the developed software

3.2. Project 1: zOmbie

The first project is called 'zOmbie' and it was carried between October and December 2003. The development team of zOmbie project involved four

students and two researchers as the development resources.

The project was given a time frame of eight weeks and the total effort was 1070 hours. The team worked 24 hours a week (i.e., four days a week, six hours a day) in the development facilities. The project was divided into six small releases having a duration of one or two weeks. Two weeks prior to project launch the team performed a self-study on XP-literature [2]. In addition, two days training on XP practices and tools was organized.

3.3. Project 2: Naked Zombie

Project called 'Roger' was carried out between October and December 2004 and was fixed to last eight weeks. The development team involved four students as the development resources. The team worked 24 hours a week in the development facilities and the total effort was 810 hours. The Roger project was divided into two subprojects: The first subproject implemented an OVM for mobile phones, because the previous version of MIDP-OVM was not mature enough for the project. This subproject took the first half (four weeks) of the Roger project.

The MIDP-OVM is application independent and during the development time of MIDP-OVM, an existing example application of the Naked Objects Framework was used to test MIDP-OVM functionality.

The second subproject, called 'Naked Zombie', took the second half (four weeks) of the Roger project and the total effort was 380 hours. Naked Zombie implemented the Stock exchange application. In this paper, only the Naked Zombie is examined. During the Naked Zombie –project, the team had access to the outcome of the zOmbie-project, including the working implementation, source code and design documentation.

One of the team members had participated in implementing the first version of MIDP-OVM prior this project, and thus had experience of Naked Objects. Two weeks prior to the project launch, the team performed a self-study by studying a basic book on XP [2] and Naked Objects [6]. Half a day training sessions were given on creating Naked Objects application and current status of the MIDP-OVM and another half a day session on the Mobile-D development rhythm, architecture design and XP practices.

Table 1 lists the software environment used in zOmbie and Naked Zombie -project.

3.3. Data Collection

Both quantitative and qualitative data were collected. Developers collected the use of effort for each defined task with a precision of 5 minutes, using paper and pen and an in-house developed tool. The team used continuous integration and the CVS repository was updated on a daily basis. After the project had ended, both development teams have been interviewed. The interviews have been recorded and transcribed.

Table 1. Technical environment

Item	zOmbie	Naked Zombie
Server side language	Java 2 SDK 1.4.1	Java 2 SDK 1.4.2
Phone side language	Java MIDP 1.0	Java MIDP 2.0
Web server	Tomcat 4.1	-
Framework	-	Naked Objects Framework 1.2.2
Database	MySQL 4.0.9 + Java connector	XML Object Store
Development environment	Eclipse 2.1, Nokia Dev. Suite 2.0 for J2ME	Eclipse 3.0.1, Eclipse ME J2ME Plug-in
Software configuration management	CVS (1.11.2); integrated to Eclipse	
Documents	MS Office XP	MS Office XP, Rational Rose

4. Results

In this section, the results of the comparative case study are presented. First the user interface layouts are compared. This is followed by development process data including effort distribution, estimation accuracy and precision, distribution of task size and user story effort and growth of the code base. Thus, the comparison will be based on a fairly large number of exploratory empirical data.

Table 2 compares the key metrics of zOmbie and Naked Zombie -projects. The calendar time was fixed before the projects.

The total effort shows the time the team has been in the development facilities. Effort used for tasks indicates the project velocity [2]. Logical Lines of Code (LLOC) [4] indicates the work size, which also includes the test code. In order to facilitate the comparison, later in the paper, the code size will be shown without the tests.

Table 2 shows that considerably less effort was used for the Naked Zombie project than for the zOmbie project. The team productivity of the Naked Zombie team was considerably higher than the zOmbie team.

Table 2. Overview of the project data

Collected data	zOmbie	Naked Zombie	Com-parison
Calendar time (weeks)	8	4	-50 %
Total effort (hours)	1073	383	-64 %
Effort used for tasks (hours)	447 (42%)	158 (41%)	-65 %
# Logical LOCs implemented	6282	3416	-46 %
Team productivity (LLOC/hour)	14,1	21,7	+54 %

4.1. User Interface

Part of the user interface of the zOmbie -application is presented in Figure 2a and Naked Zombie, respectively, in Figure 2b.

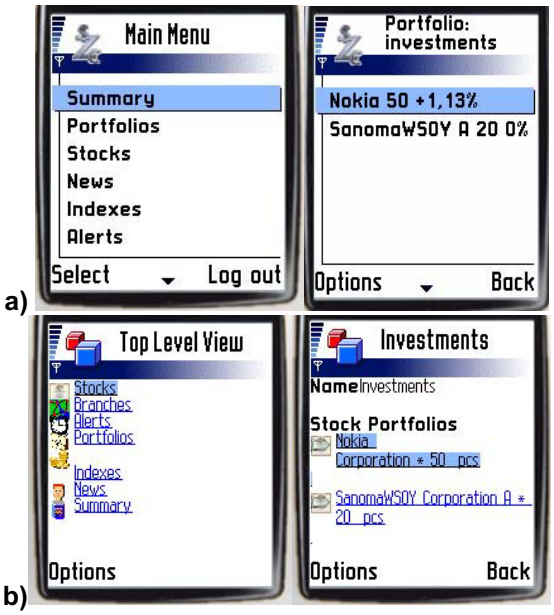

Figure 2. zOmbie user interface (a) and Naked Zombie user interface (b)

Figure 2 shows that the user interfaces are not identical but differ slightly from each other. The views and navigation are quite similar. The Largest differences are in the operations between multiple objects. The functionality of user interfaces is the same, except for three differences: The Naked Objects -version does not support multiple users – all users have access to the same objects. Built in support for multiple users was coming in the next version of the Naked Objects Framework and would considerably ease constructing this feature to application so we did not want to put effort on this. 'Trade by sending email' -feature is missing in the Naked Zombie, but that was also only partially implemented in the zOmbie project. The Naked Zombie, on the other hand, is capable of displaying actual live news, taken from public web pages. In the zOmbie application, the news are hard coded into the server for demonstration purposes.

Usability tests are not a part of this study, but there is clearly a problem in both user interfaces: They both feel slow. This is mostly due to high GPRS network latency and the fact that both applications work like a browser for the database backend - every navigation command causes a roundtrip to the server. The latency of the navigation commands in both applications is around one second when using GPRS network.

Additionally, bugs of the Naked Objects Framework caused some functions not to work properly. The Object Store sometimes returns a null pointer although a proper handle to object should be returned. Also, the performance of XML Object Store was not sufficient for frequent updates of the market information.

4.2. Effort Distribution

Effort distribution is presented in Figure 3. The coding phase consists of tasks related to the implementation of a feature. The management includes metrics collection, daily meetings and the project management work. Section 'Other' includes environment setup, studying, coaching and the documentation activities. The planning activities include planning game in the beginning of each iteration as well as architectural planning during development iterations. The quality assurance includes tasks for verifying the user stories and related tasks. The defect fixing includes refactoring and bug fixing activities. The testing includes writing test cases and pre-release testing session, which is performed prior to the release.

As shown in Figure 3, the development profiles are only slightly different in zOmbie and Naked Zombie projects. A lot of defect fixing (15 %) was done in the Naked Zombie construction phase. Much of this is explained by bug fixes required by the MIDP-OVM, which is addressed in the discussion section of this paper.

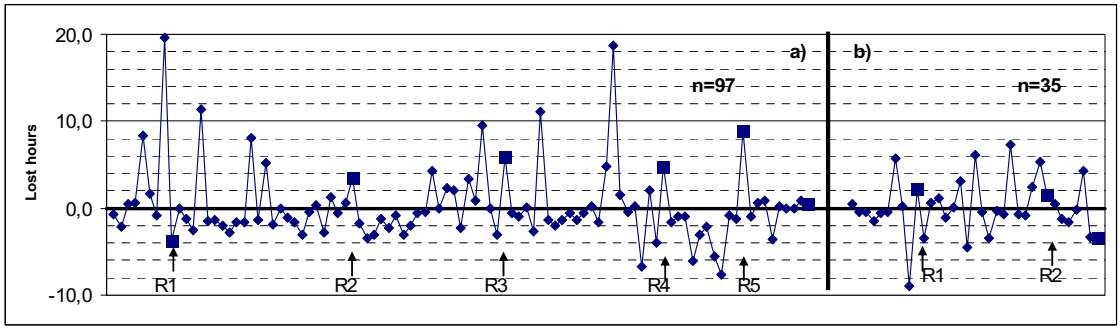

Figure 5. Hours lost by faulty estimates in zOmbie (a) and Naked Zombie (b) projects

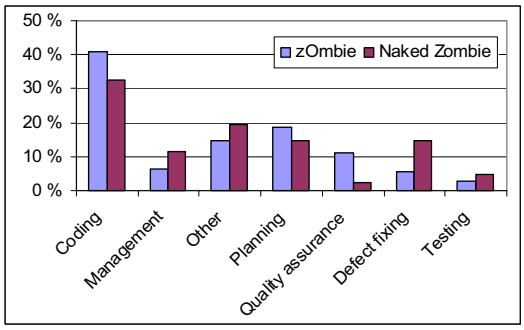

Figure 3. Effort distribution

4.3. Estimation Accuracy & Precision

Estimation accuracy is presented in Figure 4 using box plots[2]. The data used for drawing the box plots is based on the tasks that the users identified for user story level implementation. The data below the 0 % -line indicates overestimation and data above the line refers to underestimation of the tasks.

The fixed eight week timeframe of the zOmbie project resulted into six releases due to one or two week iterations. The four week time frame resulted into three releases in Naked Zombie project.

The estimation accuracy in both projects is approximately on the same level. However, the maximum estimation error tends to come down in the zOmbie project during the project, but seems to be opposite in the Naked Zombie project. Still, the absolute error range in terms of actual hours is quite tolerable. Figure 5 presents the development of the estimation precision, i.e. how many actual hours the developers lost by faulty estimates. The thick line indicates a loss of zero hours. The data points below indicate that a implementation of the task took less time than expected. The data points above the thickened line indicate that a particular task took longer than expected.

By observing Figure 5, it can be seen that the first tasks of the Naked Zombie project have a very high estimation precision but the general trend in both projects is alike.

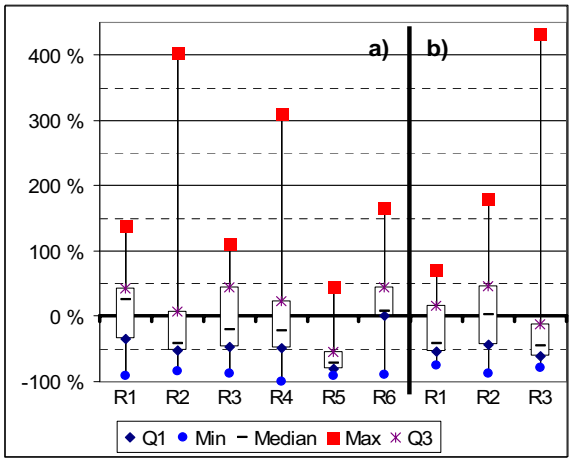

Figure 4. Estimation accuracy in terms of estimation % of zOmbie project (a) and Naked Zombie project (b)

4.4. Distribution of Task Size & User Story Effort

In the planning game, together with a customer, the team identifies the user stories to be included in the iteration. The team divides each user story into a set of tasks, preferably between 2-10 hours. The distribution of the actual task sizes are presented in Figure 6.

The zOmbie team clearly had difficulties with some tasks, but overall in both projects, the tasks were evaluated in the suggested limits.

Despite of the same functionality of both applications, the user stories are not the same and thus cannot be directly compared. However, the number of

[2] A box plot diagram visualizes the 5 number summary of a data set. Median value is the line in the shaded box area. A1 (first or lower quartile) shows the median of the lower 50% of data points. Q3 (third or upper quartile) shows the median of upper 50% of data points. The minimum value indicates the lowest and the maximum the highest values in the respective data sets.

user stories and user story effort and the speed of implementation from the customer point of view can be compared. The user story effort is presented in Table 3.

Since the functionality in both applications is approximately the same, in Table 3 we can deduce that the user stories of Naked Zombie delivered more value to the customer. Still, the implementation of the user stories took less time.

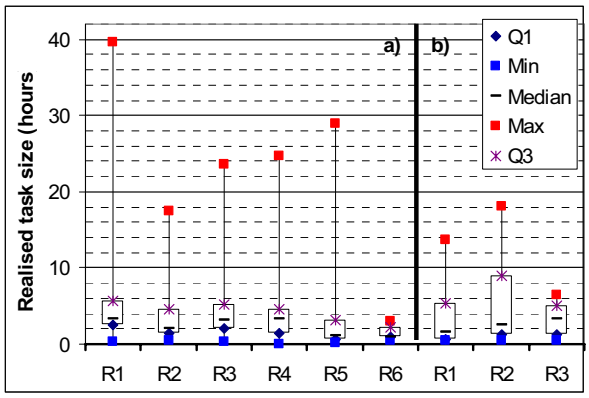

Figure 6. Actual task sizes in each release of zOmbie project (a) and Naked Zombie -project (b)

Table 3. Actual effort used in implemented user stories

	zOmbie	Naked Zombie
# User stories implemented	22	12
User story effort (median, hours)	18.9	9.7
User story effort (max, hours)	40.0	33.2

4.5. Growth of the Code Base

The development of the code base is important since it describes how the project progressed over the time in terms of actual product development. Figure 7 presents the code size development during the zOmbie and NakedZombie projects. The code sizes for each day are obtained from CVS repository after the project. The test code is left out to make the comparison fair.

The code base development is analyzed by the major software packages. This gives information on which phases of the project the development of packages took place. The architecture and structure of the software sub-packages evolve during the project. The package structures presented in Figure 7 are taken from the final version and the previous versions are adjusted to match that structure.

In Figure 7a, the 'Middlet' section represents all the code located in the mobile phone. The server consists of the server side business objects, such as 'Stock' and 'Portfolio'. Also, the database connection is handled by these classes. 'Datagen' acquires stock information updates from the web pages and feeds them to the database. The 'Servlet' package consists of only one class taking care of the HTTP-requests send by the middlet.

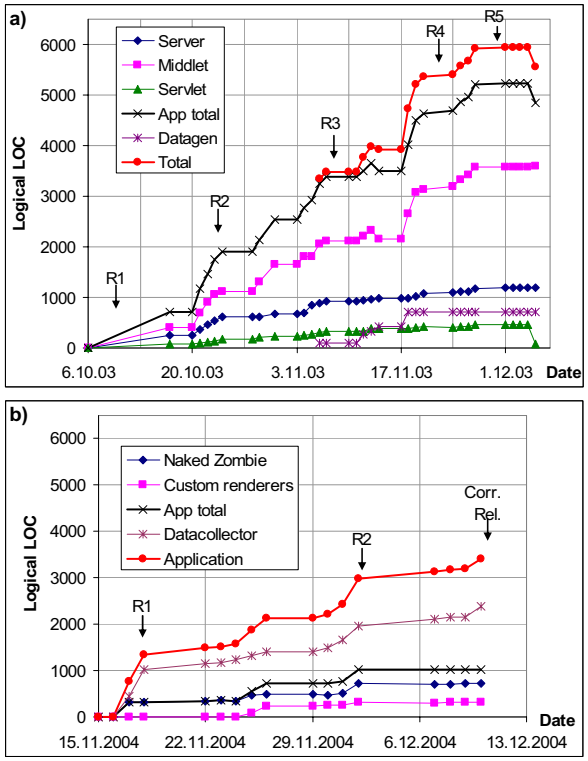

Figure 7. Development of zOmbie (a) and Naked Zombie (b) code bases

Figure 7b presents the code base development of Naked Zombie -application. Section 'Naked Zombie' consists of business objects, such as 'Stock' and 'Portfolio', defined according to the rules of the Naked Objects Framework. 'Custom renderers' are user interface components made for special cases, like displaying the history graph of the stock value development. The basic cases are handled by the MIDP-OVM and no code for Naked Zombie was needed. 'Datacollector' takes stock information from the web pages and feeds it both to the zOmbie SQL database and to the NakedZombie XML Object Store.

In Figure 7 we can observe that in the zOmbie project, the user interface code was produced from beginning of the project, while in the Naked Zombie project, the development of UI code started at the

second iteration, almost in the middle of the project. In the zOmbie project, the first signs of a web site parser appeared in the code repository in the middle of the project, while in the Naked Zombie, there was code in the third project day.

Table 4 compares the code sizes of final versions of zOmbie and Naked Zombie applications. Between the two projects, packages 'Server' and 'Naked Zombie' represent the business objects and they can be compared. 'Middlet' and 'Custom renderers' packages represent the user interface code roughly and they can be compared.

Table 4 shows significant savings in all areas except in the 'Datagen' - 'Data collector' comparison. The most significant savings (91 %) come from the biggest portion of the zOmbie project: the 'Middlet' package. Some savings also come in the business objects. This is mostly due to fact that an automatic persistence mechanism is used in Naked Zombie.

The growth of the data collector in the Naked Zombie is due to the fact that it has been redesigned and enhanced and it feeds data to both the zOmbie and the Naked Zombie applications. The data collector also contains an internal structure for representing the data acquired from web pages. Parts updating the zOmbie SQL database and Naked Objects XML Object Store are componentized and they implement the same updater interfaces. The sizes of those components are presented in Table 5.

Table 5. Naked Zombie Data collector

Component	LLOC
zOmbie updater	442
Naked Zombie updater	621
Other code (web site parsing etc.)	1328
Total	**2391**

Table 5 indicates that the real difference between the data collectors of those different persistence mechanisms is not so big.

5. Discussion

This study shows advantages of using Naked Objects for mobile applications. The greatest advantages are faster user story implementation and savings in code size.

A notable feature of the Naked Zombie project compared to the zOmbie project is that the beginning of the project goes rapidly with very precise task estimates. This provides initial evidence that creating a basic Naked Objects application is a straightforward task. Later in the project, creating the datacollector and special views then approaches more to the traditional development.

A drawback in the study is that the Naked Zombie project started right after MIDP-OVM implementation, so the project team was already in good pace and the process and tools were already familiar. This gives an advantage for the Naked Zombie project. Another issue in the research setting favoring Naked Zombie team is that the material of the zOmbie project was available to them. While this obviously facilitated the requirements gathering this does not explain the savings in the amount of actual code produced. The team reported the most value of the access to the zOmbie project outcome was a database schema design document from which the Naked Objects classes were easy to derive. These two reasons may partly explain the better team productivity presented in Table 2. Difference in productivity might be also explained by smaller team requiring less communication.

Another drawback of this study is that quality of code, which might explain difference in productivity, was not considered.

On the other hand, the fact that the zOmbie project was based on a mature technology somehow counterbalances this study. Also there were not radical changes in requirements during the project so the difference to Naked Zombie is not so great.

The study revealed that the version of the Naked Objects Framework used was not mature: There were bugs in the framework, XML Object Store was not

Table 4. Comparison of code size (test code excluded)

zOmbie			Naked Zombie			
Package	**LLOC**	**Proportion**	**Package**	**LLOC**	**Proportion**	**Comparison**
Middlet	3590	65 %	Custom renderers	310	9 %	-91 %
Server	1190	21 %	Naked Zombie	715	21 %	-40 %
Servlet	69	1 %	n/a	n/a	n/a	-100 %
App total	**4849**	**87 %**	**App total**	**1025**	**30 %**	**-79 %**
Datagen	715	13 %	Data collector	2391	70 %	+234 %
Total	**5564**	**100 %**	**Total**	**3416**	**100 %**	**-39 %**

adequate for frequent updates of stock information, the documentation was not very comprehensive, and the working examples did not cover special cases, such as building one's own database backend. All these reasons caused unnecessary struggling with Naked Object Framework and the four weeks time span did not encourage trying another alternatives as the database solution.

Authors of the Naked Objects Framework have already put effort on these issues in the upcoming framework version.

This study shows that the UI takes over half of the application code when using traditional mobile software platform. This is in line with the study done by Myers and Rosson [5]. Naked Objects bites into the fact and the greatest savings when using Naked Objects come in the user interface side, due to autogeneration of the user interface. However, the usability of the produced applications should be tested, because UI autogeneration may have effects on the usability of the applications. Also run-time properties such as CPU and memory usage need to be studied.

One of the issues affecting usability in results of both projects, is the high latency of the mobile networks. The round trip time of GPRS networks is around one second. The best way to address this problem is to redesign the client-server protocol so that the client keeps the most valuable data in a local cache and uses the network connection to synchronize data rather than waiting for the server in every navigation command.

However, despite of these challenges, there is still value in using Naked Objects in the mobile environment if the main target is a desktop environment, since the mobile application comes almost free due to the UI autogeneration. If further improvement in mobile UI is needed, a custom client and protocol could be built and Naked Objects could be used as a backend. Also, more advanced mobile networks, like WCDMA, bring down the latency to around 200 milliseconds, which reduces the need for a custom protocol.

As a conclusion, two principal lessons learned can be identified:

- This study provides confirmation that user interface development takes the largest effort of the mobile business applications.
- Naked Objects has a potential to save significantly in development effort of mobile business applications on the condition that the usability of the autogenerated UI is on a good level.

6. Conclusions

This paper presented a comparative case study where the same application was implemented in traditional mobile software platform and Naked Objects.

This study shows that when using the traditional mobile platform, over half of the development effort goes to the developing of the user interface. In this study, the Naked Objects was able to save 91 % in the user interface code and 79 % of the whole application.

The version of the Naked Objects Framework used was not considered to be mature and caused some problems during the development, namely the default database backend was not sufficient for intense database operations and no instructions how to easily build your own was not found.

The Naked Objects technology, namely user interface autogeneration has an effect to the usability of the application, but this is not within the scope of this paper.

Due to the great savings in the development effort, we feel that the Naked Objects technique is worth exploring further.

7. References

[1] Abrahamsson, P., Hanhineva, A., Hulkko, et al., "Mobile-D: An Agile Approach for Mobile Application Development", OOPSLA 2004, Poster session, Vancouver, Canada, 2004.

[2] Beck, K., *Extreme programming explained: Embrace change*, Addison-Wesley, 2003.

[3] Bettin, J., "Measuring the potential of domain-specific modeling techniques", Proceedings of OOPSLA 2002.

[4] Humphrey, W., S., "A discipline for software engineering", Reading, Mass.: Addison Wesley, 1995.

[5] Myers, B. A., Rosson, M. B., "Survey on User Interface Programming", Proc. CHI'92, ACM Press, 1992, pp. 195-202.

[6] Pawson, R., and Matthews, R., *Naked Objects*, J Wiley, 2002.

[7] Pawson, R., *Naked Objects*, PhD thesis, Trinity College, Dublin, 2004.

[8] Pawson, R., and Wade, V., "Agile Development Using Naked Objects", XP 2003, pp. 97-103.

[9] Yin, R., K., *Case Study Research Design and Methods*, 2nd ed., Sage Publications, 1994.