SCORE Project Report

Distributed Decision in a Mobile Context

Fall 2008

Department of Computer Engineering
Bogazici University
Turkey

Revision History:

Version 1.0 12/30/08 Kaan Yucer. Created

Version 1.1 02/28/09 Kaan Yucer. Edited

Preface:

This document addresses the requirements of the project. The intended audience for this document is the SCORE committee.

Target Audience:

SCORE committee

Project Members:

Kaan Yucer (CEO)

Mahmud Resid Cizmeci

Ilkay Ozan Kaya

Elif Akan

Fatma Ekici

Ali Karasu

Table of Contents

Table of Figures

ABSTRACT

Large companies with employees working at distant places face the problem of reaching group decisions. An efficient method of solving this problem is using an online voting system which can be accessed by all employees. We have developed an e-mail based voting system to provide communication among employees. System properties are set in accordance with the SCORE contest project description. The project is also set as the requirements of Software Engineering course at Boğaziçi University, Turkey. We have worked on this project using a spiral development approach. We followed the distinct phases of project development life cycle: requirements specification and analysis, design, implementation and testing and documentation. To complete this process, we followed a weekly schedule with internal milestones to track the project status. Requirements elicitation was an important part of the project, so we spent an important amount of time for inter-group communications, documentation and technology analysis and communication with our stakeholder. We focused on the validity, verification and realization of these requirements. When designing and implementing the software, our main goals were efficiency, scalability and reliability. Lastly, we tested our project for functionality and security using various test cases. We followed major software engineering processes to put theory into practice. Our instructor Ayşe Bener, our reviewers Ayşe Tosun and Aydın Ulaş and our stakeholder Stuart Faulk helped us in the development of this project with their feedbacks and suggestions. In the end, with the contributions of every team member, we implemented a software product meeting the requirements of our stakeholder.

1. INTRODUCTION

1.1. Purpose of the System

Companies with large number of employees who work at different geographical locations and travel a lot may need a system for communication. Opening polls and casting votes on them is probably the easiest way of gathering the different views of the employees on a given matter. The main purpose of the proposed software system is to provide such companies with a powerful tool to cast votes on a poll and send their own comments. The reason behind this is to speed up decision making processes and save companies from the costs of scheduling meetings with all the employees. Moreover, this would save time for the employees so that they can spend more time on their own tasks. Such a system also allows all employees to raise their voices in an open and democratic way.

In order to achieve these goals, we need a system that allows the group members to do the following tasks:

- start voting procedures
- exchange point of views
- vote
- observe the voting results

1.2. Users of the System

Two types of users have been considered for this system. The first group consists of the group members who can do the four operations that are mentioned above. The second group consists of the administrators who have the privilege to observe the current state of the system and keep track of the transactions.

The medium of information exchange is also limited. The system focuses on using e-mails as the first and most important communication medium. There will also be support for SMS since cell phones are used widely by almost all employees. But this system's scope will be limited in the sense that there will be functional prototypes that simulate SMS behavior instead of a real SMS architecture. The system can later be expanded to include real SMS functions.

1.3. Objectives and Success Criteria

The main objective of this project is to accompany the employees with an efficient and easy-to-use system to exchange their opinions and cast votes. The commands available to the users that enable connections to the system should be easily understood and easy to use by the employers. Moreover, the system should be fast and precise so that broadcasting messages, gathering votes and publishing poll results can be done effectively.

Another important aspect to consider is security. The system should be able to detect third parties and unauthorized users. Therefore there is a need to implement secure protocols.

Last but not least, scalability of the product is an important feature. Number of employees in the system can grow rapidly, or new technologies such as Wi-Fi or 3G can be supported.

1.4. Overview

The report will start with a description of the development process we have used throughout our project. In the third and fourth chapters, the problem will be stated and the requirements of the project will be given. The fifth chapter will deal with the architecture we have used in the project. The sixth and seventh chapters will describe our project and management plans respectively. Important implementation details will be supplied in the eighth chapter. The ninth chapter deals with the validation and verification of the implemented project. We will describe our outcomes and the lessons we have learned in the tenth chapter.

2. DEVELOPMENT PROCESS

Our stakeholder Stuart Faulk, our instructor Ayşe Bener and the course assistants Aydın Ulaş and Ayşe Tosun have directed us in the development process. In the software engineering course we had strict deadlines for the delivery of the documents.

As far as SCORE requirements were concerned, we were free to implement the project at our own discretion as long as we finished on schedule and meet the

requirements. This helped us to have full control over our project and made it like a real life experience. We learnt a lot especially in making critical decisions.

First we gathered and analyzed the requirements. At that stage our stakeholder helped us clarifying issues related with the requirements. Then, we designed the system considering various design alternatives.

In coding phase our strategy was to implement in small increments. So we first coded the basic e-mail sending and receiving parts. We left some detailed parts such as creating a GUI for the administrator of the system for future coding. After performing the unit tests we moved on to the more complicated parts of the coding. We also decided on some additional properties that we thought were necessary, such as a password for each user to increase security, which was an advice of our stakeholder, and a check box to the admin window to choose if the users should enter their passwords in voting or not. At the end, we processed all the test cases and corrected the implementation accordingly.

When we consider the software development life cycle methods, we can say that we have used a Spiral Process. The Spiral Process involves a close working relationship between the designer and the users. A Spiral Process begins with a simple implementation of the project requirements. Each iteration adds more functionality until the full design is realized. The lessons learned during each incremental stage of development are applied to refine the design.

The advantages of the Spiral Process [4]:

- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.

We have selected this development style since we needed to implement changes according to the feedback from our assistants and our stakeholder. They gave us direction meet the requirements in the best possible way. This spiral process enabled us to have a working prototype early and help us to set realistic goals for the project.

3. REQUIREMENTS: PROBLEM STATEMENT

Large companies with a horizontal organizational structure may need to take common decisions on certain business issues. If the employees of such companies travel around the world and work in different countries, they cannot always make plenary meetings. Company may need to take a decision urgently and organizing such meetings requires time and money. They should rather take such group decisions remotely from wherever they are, using available technological infrastructure. By this way they can reach a decision as quickly as possible and the costs of organizing meetings are eliminated.

For this problem, we need to develop a system which will allow votes to be casted and voting results to be obtained for decision making among a predefined group of people. Voters should be able to call for voting on a certain issue. This may be done using different asynchronous mechanisms and technologies such as SMS for mobile phones, e-mails sent through Wi-Fi etc. In other words, users will be able to use the technology that is available to him/her at that moment. If all of the mentioned technologies are available, he/she will use the one that he/she finds easier to use.

The system will be used by a predefined group of people (voters) and will not be available to those who are not members of the group (i.e., it is not for open or public elections). In that sense, aspects such as security should be considered. One should also note that such a way of taking decision is only suitable if the company has a horizontal structure. The main reason for this is that when a member calls for voting, the decision issue and question is sent to each member of the group and votes of every member count equally as far as their contribution to result of the decision process is considered. To sum up, we need a system to enable:

1. Inform that a decision should be taken for a given business reason
2. Broadcast each point of view to others
3. Choose or vote between given options
4. Compute, present and communicate the results of the vote

It is preferred that the system is implemented as a functional prototype or a partial implementation using a short list of technologies. In that sense the algorithms used in the

project gains importance rather than the technologies. In our system, this problem is reduced to e-mails due to the limited time we had available. Our conversations with our stakeholder eliminated the use of other technologies, such that our system works only over e-mails. There are, however, classes and functional prototypes for the SMS behavior, which makes it extendible.

4. REQUIREMENTS SPECIFICATION

The Project "Distributed decision in a mobile context" includes many requirements related to the aspects of the system that are not directly visible to the user. Therefore in requirements elicitation phase, we made our progress very carefully in order not to miss any requirement. First of all we read the Project description [1] carefully and discussed the requirements stated in that document with our stakeholder Stuart Faulk. We consulted our stakeholder whenever we needed to take decisions concerning a requirement. Mr. Faulk was helpful in writing the requirements document in the right format and in the requirements elicitation process. We have sent him e-mails with our questions about the requirements and his responses to these questions helped us to come up with the requirements.

4.1. Users of the System:

It is a good starting point to mention about the users of the system, since their needs are taken into account in the requirements elicitation. The system is going to be used by companies with horizontal structures and employees travelling around the world. Therefore, we need to have a user friendly system that will be used by those employees easily. The system should be helpful when the user makes mistakes while opening a poll or voting for a poll. Moreover, the system is going to work in a mobile context to be able to reach the members that are located far away from each other.

Although we are dealing with horizontally structured companies, we thought that for such a voting system, it is indispensable to give some people administrative rights. We consulted our stakeholder and he approved the necessity of administrators. Therefore

two types of users have been considered for this system. The first group consists of the group members who can do the five operations below:

- Send an alert message informing that a decision regarding a certain matter is required from all members of the group by opening a poll.
- Vote using the preferred means of communication
- Send opinions about the voting questions and choices
- Close the poll for some reasons (that can only be done by the poll opener)
- Receive the results of a decision process

The second group consists of the administrators who have the privilege to add/edit users to system, delete users from system, and observe the current state of the decision processes in addition to the five operations above.

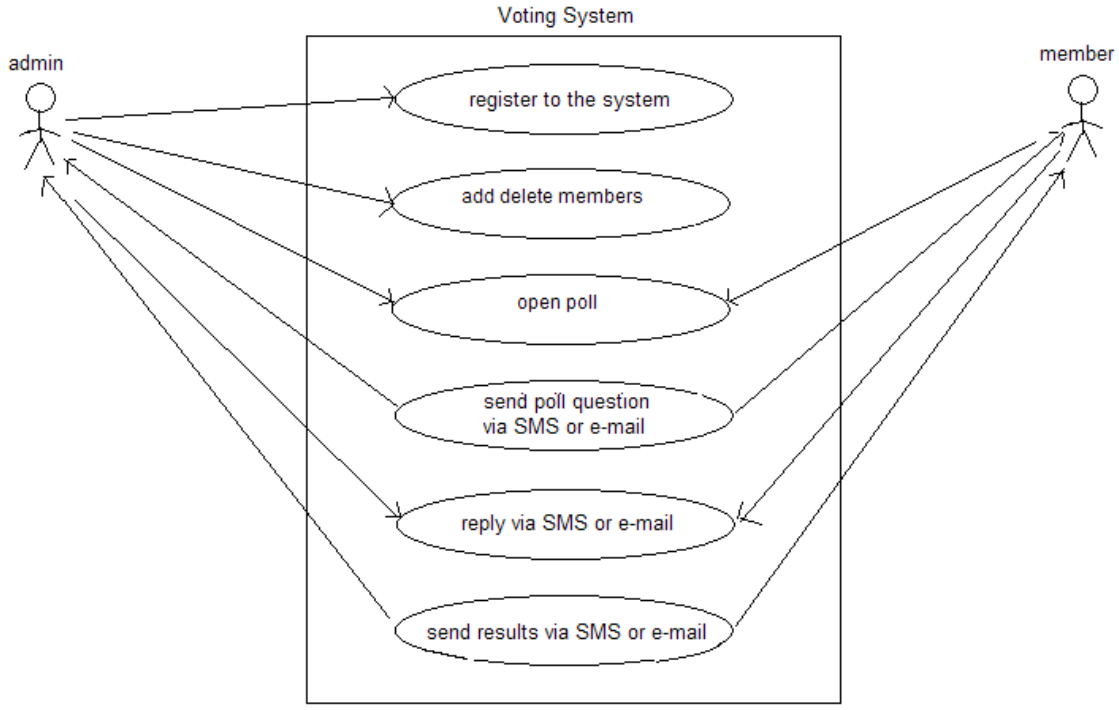

Figure1. Use Cases

4.2. Poll Types

Another important requirement other than the mentioned above is that the decision taking will be in the format of a multiple choice question. That means when a user calls for a decision process he/she opens a new poll. This poll can be constructed in two formats: “single choice” and “multiple choices”.

- Single Choice: A user sends his/her question and multiple choices including the type of the poll as single preference. In this type of poll a voter selects only one of his/her preferences among the multiple choices sent with the question. For instance, for a poll question with 5 choices from ‘a’ to ‘e’, the user has to select only one of the choices (could be c) and cannot vote for two choices at the same time.
- Multiple Choices: A user can send all of his/her preferences out of the multiple choices. For instance, for a poll question with 5 choices from ‘a’ to ‘e’, the user may select any combination of the choices (could be a,c,d). This requirement is mentioned in the description of the Project as well as in the responses of our stakeholder.

4.3. Other Requirements

- The system will be accessed using e-mails. These e-mails should be in a specific format according to the type of the poll being opened or being voted for.
- The system should alert all members of the group and inform them when a new poll is created. This will be via e-mails and in the suitable format.
- The system should alert members by sending messages to them. They will be sent messages using their preferred communications methods. As stated before, we consider e-mails for polls. Nevertheless we will provide necessary classes and prototypes for introducing the SMS technologies to the system.
- Additional information about the poll results will be supplied to the people that are connected via e-mail. The additional information will be of the type chart graph showing the percentages of votes received for each choice. Also the chart attachments to e-mails will be in the jpeg format, which is considerably small in size.
- The system should call the members of the group for the voting, send the necessary information, and then it will start the voting process. We do not need to process ACK’s starting the voting process.

- Voting will always be anonymous, i.e. when a user votes for a poll, the system will know the sender, but other voters won't, since this vote won't be broadcast. People who want to share their views can send their opinions about a specific poll in a separate e-mail.
- Users will broadcast their opinions about the vote during the voting process. The number of these messages is not fixed.
- Each member should get one vote per poll.
- The system should cope with mistakenly sent e-mails. Evaluating the votes and informing all members of the group is another requirement for the system. The voting procedure comes to a closure after a certain time has elapsed or when all members have voted or one of the choices has assured to be the winner.
- The poll owner should be able to stop the poll if necessary. In this case, there should be no winner. The voters should be informed and, if desired, a vote would have to be initiated.
- For all polls, a quorum should be defined in advance. A "quorum" is the smallest number of votes needed for the poll to count.
- If the deadline expires before a quorum is reached, the vote does not count. The quorum can be changed by the admin when the system is online. This feature guarantees that the decision is made by the majority of the voters.

5. ARCHITECTURAL DESIGN

Our design is a three-tier architecture. These are database, business and presentation tiers. We thought that it is better to implement the database and the user interface parts independently. The middle tier (business tier) is responsible for doing processes and providing communication between database tier and presentation tier. This architecture eases our implementation and provides a clear medium to make changes in the user part independent of the background processes and vice versa.

In our design, business tier talks to the database, processes data and provides information for the presentation tier. Presentation tier communicates with the user, gets data from the user and sends data to the user.

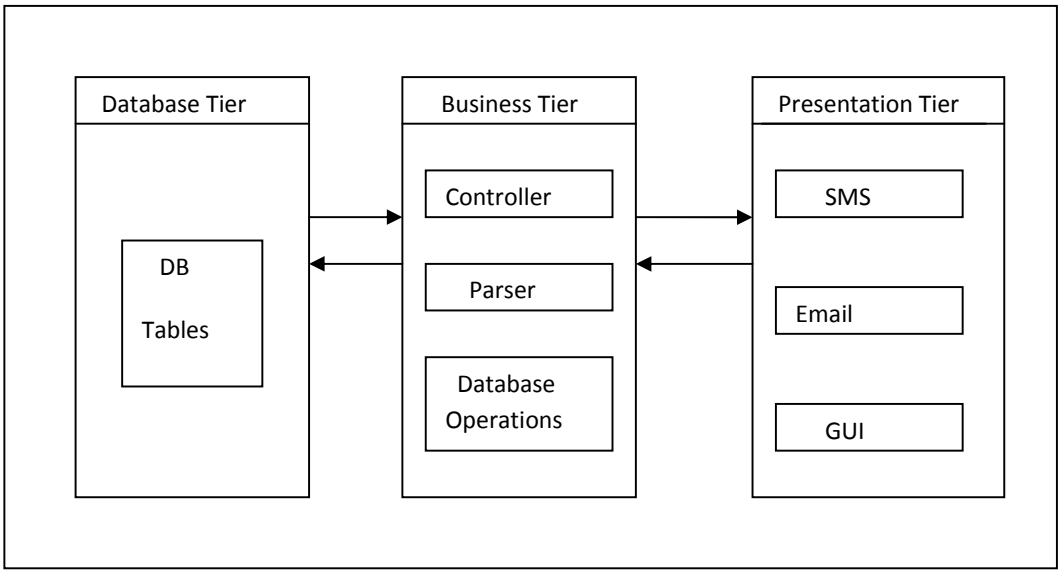

Figure 2. Communication of tiers to each other

5.1. Database Tier

All necessary data is stored in our database: information about the users, information about the poll questions, information about the poll choice scores, records for all users for each poll and results of all closed polls.

In the first version of our design, we decided to store the data specified above. Since our project development method is a spiral method, we decided to store also an error log to keep track of the errors that the system produces. There was also need to store system information, since the system will be used by different firms with different e-mail addresses, system administrators for voting. That's why this data is needed to be kept in the database and restored when the system is started.

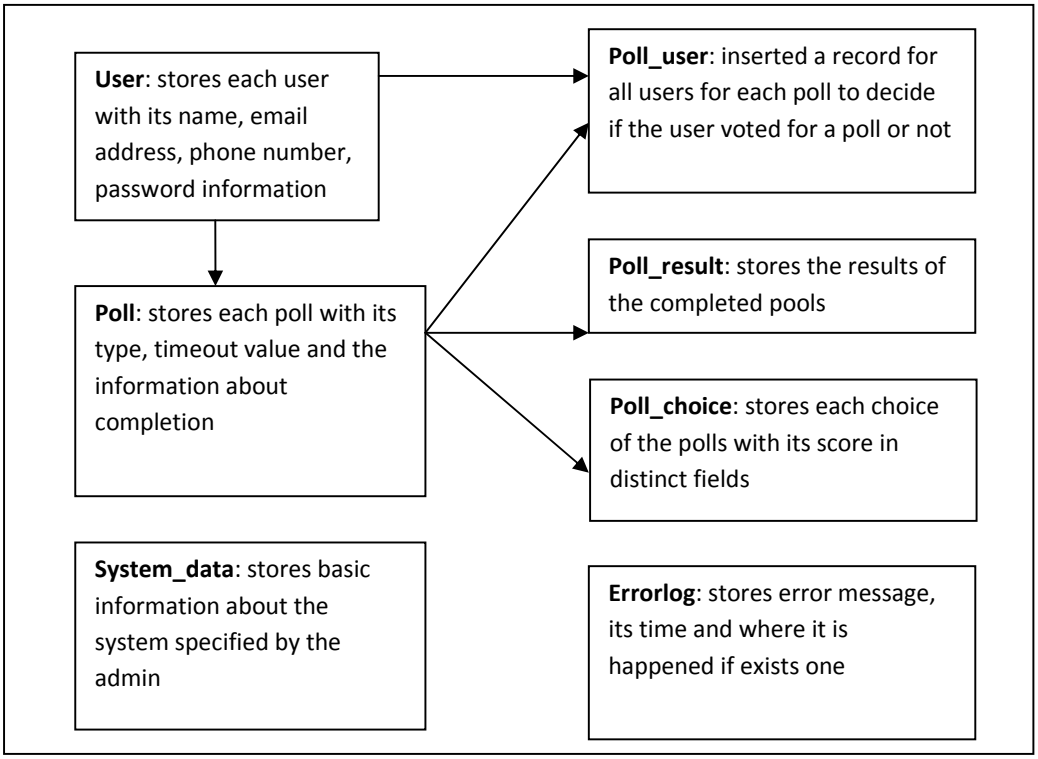

Figure 3. A simple diagram about database tables

5.2. Business Tier

In this tier we have three main parts

1. The parser
2. The controller
3. The database operations

Apart from the system admin all of the users communicate with our system via email. Hence, we need to parse the data coming from the users. When an email arrives, the parser is called to parses the string in the e-mail and then, it calls the necessary functions from the database operations. SMS agent is also designed by the same logic that a future extension of our software product could be to implement SMS similar to e-mail.

The controller is the most important part of our project. Controller runs continuously as soon as the system is online, only stops if the admin stops the system. It communicates with all other parts and makes the necessary processes when needed. It

calls parser when an email comes, it prompts the email agent when an email is needed to be sent, and it calls relevant database operations.

All the connections to database are done in the database operations part. When an insertion, selection, deletion or update operation is needed by other parts they directly call functions in the database tier to make the changes.

5.3. Presentation Tier

All communication with users is done in the presentation tier. This part consists of three modules:

1. SMS module
2. Email module
3. GUI module

SMS is not implemented in our project since we needed to purchase bulk SMS Messages from a GSM operator. We are asked by our stakeholder to prepare a prototype of the SMS agent which would be enough, so we prepared the classes to add the functionality for voting via SMS to the project. For the future work, all we have to do is implementing the functions created in the prototype and buy bulk SMS from an operator.

For the email module, we decided on use gmails. An email account would be enough for our purposes, to mainly send and receive small sized e-mails. The user sends her email to that email address and gets the questions, results or any acknowledgements via that email address. The formats of the emails are specified; the parser works through this format and acknowledges the user if an email is sent with a wrong format. There are formats for the following: Opening a poll, voting for a poll, sending an opinion about a poll and closing a poll (by the poll opener).

In the GUI module, an interface is created for the administrator. The admin can start and stop the system using the buttons in this window. The admin can add or edit members to/from the voting system. There is also a configuration tab in this GUI. With that tab, the admin can specify some parameters such as the sleep time (how much the system will wait to check for e-mails and closed polls), the “password required” checkbox, which specifies if the user should add his/her password when voting for a poll,

opening a poll or sending an opinion about the poll. The e-mail of the system, the password of this e-mail, the SMS server and its password can also be set in this GUI by the admin. In the last tab, we have history information about the closed polls. While the users get information about a closed poll spontaneously, the admin can see the information about all the closed polls in the database if he/she wants to.

Also some other additional modules are included in the project to create the security keys (the key generator part) and to create the pie charts about the results to be sent to the voters. These are used by presentation tier and business tier when necessary.

6. PROJECT PLAN

Our project plan depended mainly on the tight deadlines assigned by the course assistants. We have also agreed with our stakeholder on these deadlines so that we had a clear sight of what to complete when. These deadlines mainly dealt with some documentation about the project and we had a final deadline for the working implementation of the project. These can be seen on the schedule below:

- **5 December 2008:** Requirements Analysis Document, Coding and Security Standards Document
- **15 December 2008:** Object Design Description, Database Design, Test Plan Documents
- **30 December 2008:** Final Runnable Project, Test Results Document, User Guide and Installation Manual, Updated RAD/ ODD/ DB Design (if necessary)

After these deadlines were set, we had to find a way of preparing the code parallel to the documentation and that's why we set our own milestones in the project. These milestones mainly dealt with the design, implementation and testing of the software, which was left to us to decide on. It is important to note that these milestones were formed only for internal use in the project group and did not depend on the expectations of our stakeholder or the course assistants. We set some intermediary goals in order to be able to come up with the working project on the specified deadline above.

The first phase in our project was to talk about the requirements of the project. We needed to have a rough idea of what is needed of the project, so we gathered and discussed about the project description. After this step, we came up with some questions about the requirements for our stakeholder so that we could have an idea of his expectations. The responses made some vague points clear: We did not have to buy bulk SMS from an operator and technologies like Wi-Fi or 3g were out of our scope. We also had new requirements like “quorum”, which is the minimum percentage of the people needed to vote such that the poll results can be valid. After this feedback, we reviewed and revised our requirements and started with our internal milestones. Every milestone was completed in one or two weeks. Since we used an iterative approach, every step answered some requirements and included some design, implementation and testing. These testing, in some way, ensured that we would not have unexpected failures as we get closer to the delivery time.

Our first milestone was to code a simple program with the capabilities of sending and receiving e-mails. This was an important step to consider, since the system we are required to build mainly works through e-mails. The trial version of the Jscape library [2] helped us in the POP3 and SMTP connections. We registered for a gmail account, which would serve as the basis e-mail address for our project, and tested the code whether we could send and receive e-mails using this e-mail address. The completion of this step meant that we could move on to understanding the contents of the incoming e-mails. We also included some functional prototypes for SMS behaviors, but did not implement them in detail since our stakeholder only asked for functional prototypes as opposed to a working SMS architecture.

The second internal milestone was to develop a parser for the project. This parser should get the contents of the e-mails and parse it according to the predefined e-mail formats. We agreed on e-mail formats for opening polls, voting, sending opinions and closing polls by the poll opener. The parser should be able to identify these and send acknowledgements to the mail sender accordingly. The parser was designed and implemented. E-mail formats were tested to see whether the parser worked correctly and relevant messages were sent back to the users.

The third phase involves an important design issue: the database of the project. The database should keep information about the users (e-mail, password, name etc.), about the open polls (choices, type of poll, poll id), about security codes for each user for each open poll, and about the votes (which choice of which poll has how many votes). It also kept information about the system (mail address and password, SMS provider and password etc.) and, though not expected by our stakeholder, the results of the closed polls. The database was integrated into our mailing and parsing system. The testing mainly dealt with the necessary queries.

As part of the fourth milestone, we implemented a fully working voting system with the necessary security measures. The system realized the requirements listed in the requirements document we prepared before. Users could open polls, vote for polls, send opinions for polls and close open polls (this last option can be done only by the poll opener). The application logic, which enabled the score calculations, quorum and timeout checks, was also implemented. So, when a voting ended, the results were sent to the voters. The pie chart, which is sent to the voters as part of the results, was implemented with the help of a library. Password and randomly generated security keys were checked in e-mails. The testing involved the first three parts described in the "Validation and Verification" part, which deal with the functionality of voting, security measures and human errors/extreme cases. In this step, we also decided not to track the connections of the voters at each time. The reason is that our system works only over e-mails and if new connection types are added later, this step can easily be added to the controller class without much effort. So, we crossed it out for the first product since it wouldn't have much meaning

The last step of the coding phase was to build a GUI for the administrator. In this last phase, we had to work overtime to meet the deadline. This GUI enabled the system parameters like e-mail address, password, SMS server and password to be configured dynamically. The quorum rate and the sleep time for the system between checking for e-mails were also designed to be easily configurable through the admin screen. We added a history tab for accessing the results of the closed polls. Adding/editing/removing user screen were also added. After the implementation, we tested this module and fixed the

bugs. Since the design/coding/testing phases were now over, we only had to make a full test of the working system to check whether there are any faults, errors or bugs in the software product. It is true that we tested every part we implemented, but this doesn't necessarily mean that they are working now. The new parts we added to the project might have affected the old parts which mean that we have to check the functionality of the whole system for every case listed in the Validation and Verification section.

After these milestones were reached, we set up a testing date, where we tested all the cases described in the "Validation and Verification" section. When we were satisfied with the results of our testing (all cases were working correctly) and could not find any more bugs to fix, we ended this last, and most important, phase of our development. We had a working project with the necessary documents. By the completion of the last milestone, we were able to meet the project deadline.

7. MANAGEMENT PLAN

In our group, we have assigned tasks and responsibilities based on skills and experience of each team member. Below are the individual responsibilities:

Kaan Yucer – Group leader, has overall responsibility of the project

Mahmud Resid Cizmeci – responsible for coding and finding external libraries for integration

Ilkay Ozan Kaya – responsible for coding and testing

Elif Akan – responsible for the database and implementing its functions

Fatma Ekici – responsible for documentation

Ali Karasu – responsible for the group's web services

Our stakeholders can be seen below:

Assist. Prof. Ayşe Bener – our instructor, project reviewer

Aydın Ulaş and **Ayşe Tosun** – course assistants, reviewed our documents and project

Stuart Faulk – our main stakeholder assigned by SCORE

One of the most important decisions taken in the group is to create a mailing group for internal communication. The reason was to be able to discuss important issues over e-mails, since we all lived in distant locations in Istanbul and communication was a

problem for us. Moreover, this mail group helped us for brainstorming about how we should implement the necessary parts of the project and how we should document the changes and/or additions to our project in the best way. Since the group was formed of people with different abilities, these e-mails benefited us so that we could get the best possible ideas of our group of six people. Our different strengths complemented each other.

Every Thursday, we held meetings at school to track weekly progress and discuss for the next week's activities. In those meetings, we set our coding and documentation goals for that week and what we should test after the implementation. After these meetings, we created meeting minutes for all decisions that we collectively made.

The duties of the coding and documentation groups were as follows: The coders were responsible for the upcoming milestones throughout the project lifecycle. The testers were responsible for testing the up-to-date project with the related test cases. The documentation group for that week was responsible to note the changes and improvements of the design, implementation and testing. The documents described in the project plan were revised every week by the documentation group of that week. We also tried to balance the work of each group member. That's why we did not have strict borders between the documenters, coders and testers. Everybody served every area so that an equal division of labor could be reached among the group members.

Once in every two weeks, we gathered for testing the implementation with all group members. We went over the implementations and documents together so that everybody was up to date about the state of the project. These meetings served for integrating the code parts by different people, for comparing the documents and the codes. That way, everybody was aware of the situation of the project and they could easily adapt to changes.

8. IMPLEMENTATION

When implementing the code of the project, we used some open source libraries and some methodologies that are worth explaining here. The libraries mainly dealt with advanced connections and graphics, whereas the methodologies dealt with the logical

base of the project. We also used some decision processes that the project uses in order to adapt to the environment or make logical inferences using the knowledge and experiences from the environment. These libraries and methodologies will be analyzed more deeply in this section.

We have used two libraries for different purposes. One of them is the trial version of the Jscape library, which can be purchased online [2]. The reason for using this library is that it offers secure SMTP and POP3 connections with mail servers. Since security is an important criterion in our software project, we decided on using these libraries. If the software is to be used for financial purposes, this library and the source codes can be purchased online. This library enabled us to implement the e-mails in a simple way. The security it offers is also an important factor why we are using these libraries.

The second library we have used is an open source library dealing with charts and graphics [3]. We have used functions of this library to build pie charts which are sent as attachments to the e-mails to present the results of the polls. Additional information in jpeg format was a requirement for our project and that's why we needed some kind of drawing tools. This library offers simple functions that can be used for drawing purposes which we used in our project.

Some implementation and design choices we have used are also important to note in this section. One of them is the way we keep the e-mails in our code. We have used JAVA String vectors for this purpose. These vectors contain the subjects and the bodies of the incoming and outgoing e-mails. Their resize ability helped us when we were trying to send and receive a changing number of e-mails. We have also used three classes which serve as interfaces between other classes. They keep information about e-mails, polls and votes, which is helpful for exchanging large amounts of data between classes.

Another important part of the implementation is the database. We needed a system that responds in less than 0.01 seconds after receiving the votes. Since sending and receiving e-mails take a long time, we could not afford the system to be slow in another part. Keeping information about the polls, votes and users in the database helps us to respond quickly. Moreover, using databases has another advantage that its contents are not deleted when the system goes offline. This is an important criterion because for

some reason, the system may go offline and in such a case, we don't want old information about the users, about the system and about the open and closed poll to get lost. That's why we have used databases that keep important information about the voting system for increased reliability.

There are also important logical calculations in the implementation. One of them is the decision process when to close a poll. When a new vote is inserted into a poll, we check to see whether one of the choices has guaranteed to be top vote getter. This is calculated as follows: We count the number of votes for the top two vote getters and we get the number of users that have not yet voted for this poll. If the difference between the top two vote getters is larger than the number of user that are yet to vote, the poll is closed and the results are sent. We have used queries from the database to do such calculations. If the timeout of a poll is reached, we calculate whether the quorum is reached by extracting relevant information from the database. If quorum is not reached, the results are not calculated. These logical inferences using the data in the database helps the software in deciding when to stop and what to do in certain circumstances about the poll closings.

Another important implementation detail is the back-off algorithm. The system sleeps between checking for e-mails, which can be dynamically configured by the administrator. If the system keeps checking for e-mails and returns empty handed, then it should understand that there isn't much conversation going on in the voting system. In such cases, we have decided to sleep the system for a longer duration of time. If there are no e-mails 10 times in a row, then the sleep time is increased such that the program does not use bandwidth for connecting to the mail server, which most likely will result in a "no new e-mails" situation. Moreover, some mail suppliers understand that a machine is trying to connect, and if the time periods between these two connections are too small, then these servers ban that accessing software for some time, which is not desirable. In order to increase efficiency in not using unnecessary bandwidth and to wipe out the possibility of getting banned, we have implemented such a back off algorithm.

Last but not least, we should explain the security in our system. Every user in the system has his/her unique password. In addition, we use PIN's to enhance the security.

When a new poll is opened, every user receives an e-mail with the poll question, the choices and a randomly generated PIN number. If the user cannot enter this PIN correctly, the vote does not count. When the PIN is entered wrongly three times in a row for a poll, the user is banned from that poll, but not from the other polls. The reason for this second level of security is the fact that the sender address of an e-mail can be changed quite easily. Someone can learn the password of another user. In order to avoid fraud, we need to be sure that the user sending the e-mail also received the e-mail of the system, i.e. is the intended receiver. That's why there is a two level security.

We should also note that the implementation of the software was done using a spiral methodology. Since we had to finish this project in a very limited time, we could not afford to come up with a detailed design and then start the implementation phase. We started the implementation right away and then added new parts to it. This approach also helped us when there was need to change parts in the design, which was a result of the feedback from our stakeholder and our course instructor. We should also note that communication with our stakeholder was limited to e-mails, so we should not lose precious time while we were waiting for answers about design details. Last but not least, using the spiral method reduced the risk of overall failure. We could test our code after each step which helped us to find the errors earlier and not to build the entire project on wrong design choices.

During the project, we have designed and implemented some parts, then checked whether these parts worked correctly, and only then added new functionality to the project. This enabled us to deal with faulty cases more easily and we could adapt to changing expectations and requirements. This also helped us in implementation. We did not have a huge system to implement in a limited time, but we separated it into small pieces, which resulted in a more efficient outcome. We were not discouraged by the work, the small steps taken increased our motivation and we could see the results of our code directly. We can sum it up by saying that we took small steps in climbing the hill.

9. VALIDATION AND VERIFICATION

In this part, we made several tests about requirements after the coding was complete. We made our tests step by step according to the spiral process. After each step was completed, the necessary testing was done on that part. When the project was completed, we once again went over all the test cases to see whether the system acted correctly upon these cases. This was important since intermediate steps may have included some new bugs, which should be searched for and corrected.

The scope of our testing is:

- Functionality of the Software
- Dealing with Errors
- Security Issues
- Administrative Testing

Our test plan also follows the spiral methodology and can be described as unit and integration testing. Before each implementation step, we checked our design to see any logical errors. After that, we implemented that part and did unit testing (i.e. we implemented the GUI and tested only its functions). After that step, we integrated that last part into the working system and did integration testing. In the integration testing phase, we checked whether the added part worked in harmony with the already working code. Moreover, we checked whether this new part did the right thing (i.e. "add user" adds a user to the system). After correcting the buggy statements in these phases, we moved on to the next design and implementation phase. Here are the scenarios we have used in the end to test the software we have developed:

Scenario 1:

- The Administrator enters user information using the add user screen. Then, the e-mail of one of the users is changed. When all user information is entered to the system, the admin starts it. (Administrative Testing)
- One of the system users sends an e-mail with an invalid format. He/she receives an e-mail that informs him/her that there was an error in the e-mail sent. The possible e-mail formats are also sent using that e-mail. (Dealing with Errors)

- This user then uses the correct format to open the following poll, which is a multiple choices poll with timeout 5 hours (Functionality of the Software):

open poll

multiple timeout 5 h Which teams are championship contenders this year? Barcelona, Real Madrid, Sevilla, Valencia

- One of the users of the system sends an opinion about this poll using the following format (Functionality of the Software):

opinion

poll_id Why is Villarreal not listed?

- The poll opener then closes the poll using the following format (Functionality of the Software):

close poll

poll_id I'm closing this poll since I forgot to include Villareal

- The user opens a new poll with the predefined format (Functionality of the Software):

open poll

multiple timeout 5 h Which teams are championship contenders this year? Barcelona, Real Madrid, Sevilla, Valencia, Villarreal

- One of the voters tries to vote for one of the choices twice (duplicate vote of the form a, a). He/she receives an acknowledgement that such a vote does not count (Dealing with Errors).

- The users vote according to their preferences and the voting comes to an end after everybody votes. The results are sent to the users (Functionality of the Software).

Scenario 2:

- A user of the system opens the following poll which is single choice and has the default timeout value (Functionality of the Software):

open poll

single How about a surprise party at Mr. Jack's Birthday? Yes, No, Maybe

- One of the users votes for the choice d, which is not listed. He/she receives an acknowledgement that there is not such a choice for this poll. (Dealing with Errors)

- A user enters his/her security number wrong three times in a row. He/she receives an e-mail informing her that he/she is banned from that poll for making the same security mistake three times in a row. (Security Issues)

- One of the users tries to vote using a wrong password. He/she gets a warning that his/her password is wrong. The vote does not count (Security Issues).
- Everybody votes for 'Yes'. After the half of the voters have voted 'Yes' for this poll, the poll is closed since this choice has guaranteed to be the top vote getter. The results are calculated and sent to the users (Functionality of the Software).

Scenario 3:

- A member opens the following poll (Functionality of the Software):

open poll

single timeout 2 m Are you satisfied with the food in the cafeteria? Yes, No

- One of the users votes for 'Yes'. He/she then tries to send another vote for the same choice, but gets a warning that he/she has already voted for that poll (Security Issues and Dealing with Errors).
- The voting comes to an end after two minutes. In two minutes, the quorum is not reached; hence this poll is not valid. The results are not calculated and the users are informed that the quorum was not reached (Functionality of the Software).
- The user again opens a poll, but with a higher timeout value (Functionality of the Software):

open poll

single timeout 45 m Are you satisfied with the food in the cafeteria? Yes, No

- In 45 minutes, 4 of 5 users vote for the poll. In the end, when timeout is reached, we see that there are enough votes (higher than quorum) such that the poll is valid. Results are calculated and sent to the users (Functionality of the Software).

During the testing phase, we tried to catch as many bugs as possible with the above scenarios. They mainly deal with the functionality of the program: opening polls (single choice or multiple choices), voting for these polls, sending opinions about these polls and result calculations; with possible erroneous cases like e-mail errors, internet connection failures or human errors like voting more than once; with security issues like passwords, security numbers, banning users; and with the GUI, which is the control place for the administrator. We also made some verification and validation during the coding process:

Commit and rollback – since the system may crash because of some unpredictable events, we wanted to prevent the database to crash in the middle of an action. There may be wrong information in the database in case of an error during insertion or update if some of the data is inserted or updated in the system and some are left. So, we used auto commit method to prevent situations like that. The system executes all the needed queries, and if the end of the function can be reached safely, these queries are committed. If an error occurs during this process, we roll the transactions back.
Error log – in the implementation of the project, we used try catch blocks as usual. But it is better to store the errors that occur in the system for future reference and to understand the nature of the faults. So we added some queries into the try catch blocks in order to insert the error messages and time of these errors into the database.

Our purpose was to minimize the number of possible errors during the coding process and to maximize the number of bugs found in the testing phase.

In order to measure the quality and complexity of the source code, we have used Prest [5] which is a tool developed by SoftLab at Bogazici University. Thanks to Prest, we examined the static code metrics (i.e. lines of code, number of operands and operators, cyclomatic complexity etc.) extracted from the source code and compared them with the accepted standard values. In large organizations such as NASA, the codes that are passing the standard threshold values are reviewed again [6]. But we have seen that all the metric values of our source code are within the standards. For instance, the average cyclomatic complexity of the methods is approximately 10 (standard maximum value is 10). Although the complexity is close to the standard max, number of operands and operators show that we didn't use too much vocabulary in the methods. Unique operand limit is 10 and operator limit is 15; however our metrics even didn't reach the minimum values. Similarly, difficulty of the code is 4 when the standard max is 35. Moreover, the lines of code is 1/10 of the standard value (300) per method. All in all, although the complexity of the source code seems high, the modularity and readability is preserved.

10. OUTCOMES AND LESSONS LEARNED

10.1. Outcomes

Our aim in designing "Distributed Decision in a Mobile Context" project was to obtain a flexible and easy-to-understand software system which provides horizontally structured companies with a powerful tool to cast votes on a poll and arrive at a common decision. Our design had to be flexible since some parts of the design were just functional prototypes and they had to be compatible with real commercial system interfaces.

Although the recommended time period for developing this Project is 5-8 months, we had only 2 months to deliver a complete product. The reason is that we made this Project as a part of a Software Engineering course for fall semester, which needs to be finished at the end of Fall 2008-2009 semester. Actually, our semester is 12 weeks long and the project assignments were done during the fourth week of the semester. In two months, we were able to develop a system in Java by meeting all functional requirements given by our stakeholder. As per our stakeholder's guidance, we did not implement SMS module since it required bulk purchase of SMS messages. However, we have a Java class named "SmsAgent.java" which keeps the prototypes for sending SMS to the users and receiving SMS from them. If implemented in the future, it will mainly work like the SmtpGmail class, which is used for sending and receiving e-mails. This SmtpGmail class uses Jscape packages. If the project is to be extended to support SMS, the only thing that the developer should do is to put the relevant code in the body of send() and receive() methods. Below are simple examples:

Format of open poll mail:
mail subject: open poll
mail body: <password> <vote_type> [timeout value unit] <question>?
<choice1>,<choice2>, ... <choiceN>
Example :
mail subject: open poll
mail body: 1234 single timeout 3 h Which one do you prefer? pizza,burger,salad

Format of voting mail:
mail subject: <poll_id>

mail body: <password> <key> <choice>
Example :
mail subject: 1907
mail body: k3zy OzN7 burger

Format of opinion mail:
mail subject: opinion
mail body: <password> <poll_id> <opinion>
Example :
mail subject: opinion
mail body: k3zy 13 Don't you have chicken?

Examples from Administrative Tool:

Administrative users have privileged rights such as adding users, editing users and monitoring the previously finished polls and currently running polls. Examples from the admin GUI can be found below.

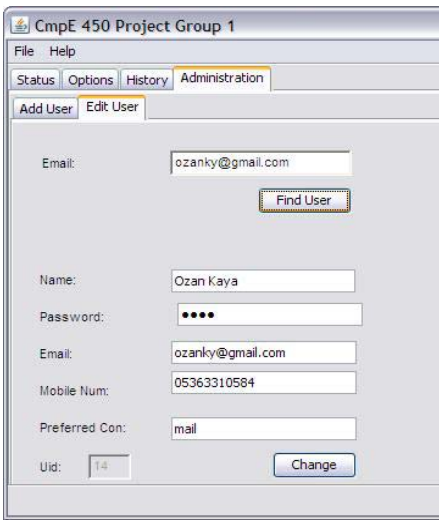

Figure 4. Admin GUI – Edit User Screen

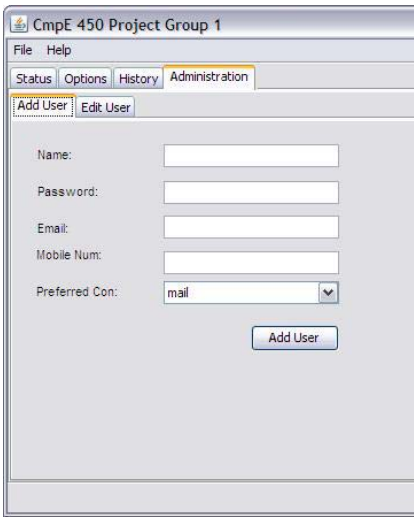

Figure 5. Admin GUI – Add User Screen

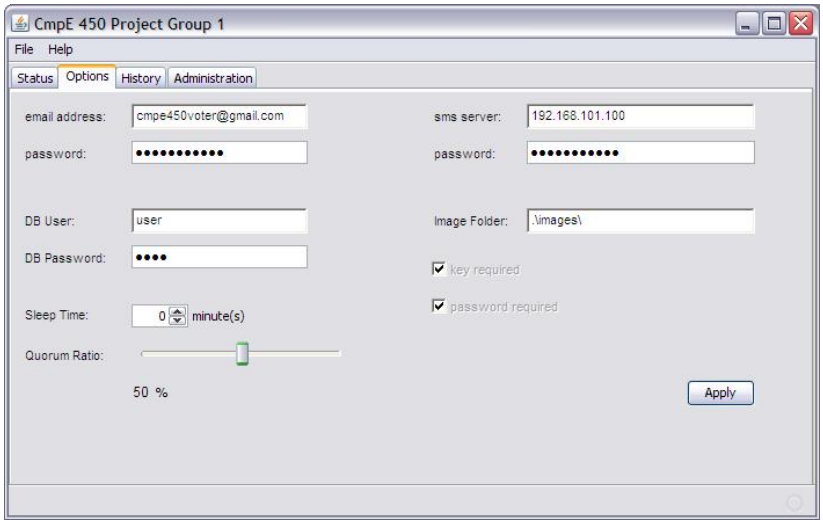

Figure 6. Admin GUI – Options Screen

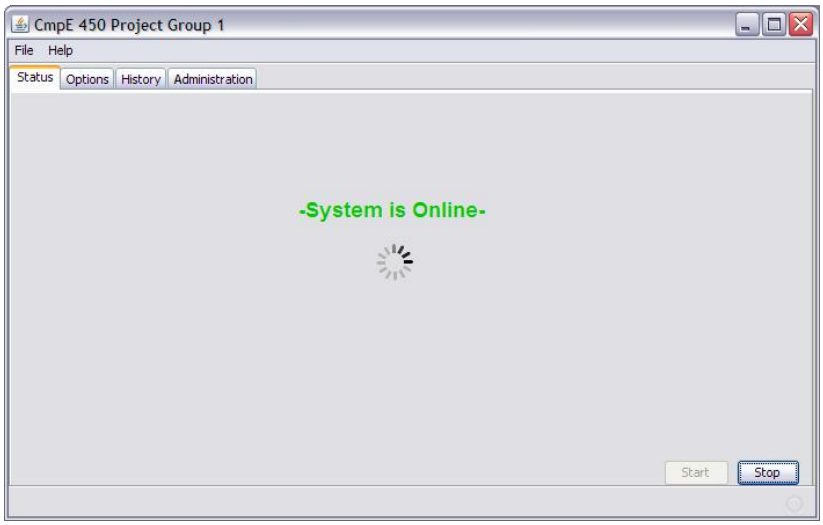

Figure 7. The System is Online

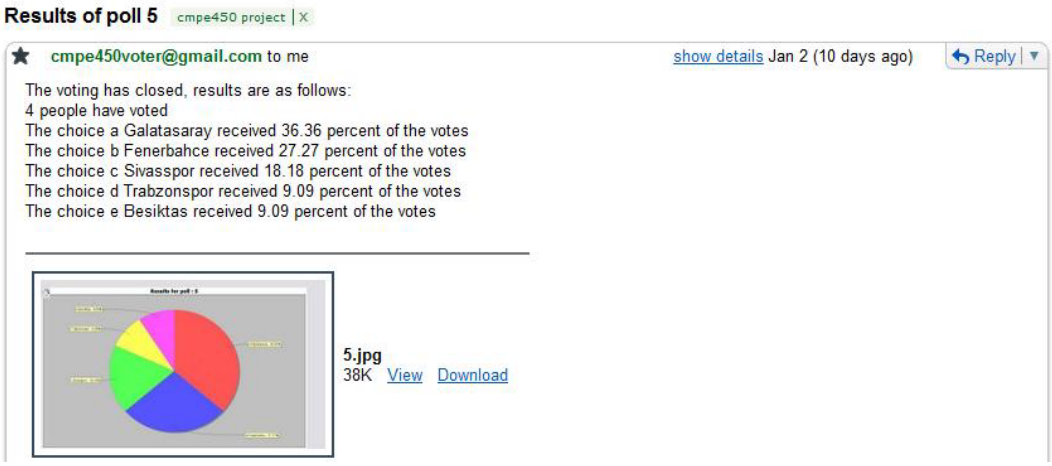

Figure 8. Email containing Poll Results

10.2. Lessons Learned

First of all, we learned working as a team and making use of Software Engineering techniques and processes in object oriented design and project management. Since we used Java in our project, it was difficult to determine classes and share the workload among the team members. While a completely object oriented design was desirable for encapsulation and readability, an increase on the failure risk and coding cost was inevitable. Since different people were writing different classes, making them compatible triggered additional coding issues and caused subtle bugs. Also, the communication costs of these people increased the overall cost of project.

Secondly, it was our first experience of a project with strict security requirements. Since this project involves communication over insecure channels, security is very important and is an indicator of how thorough software has been made. Adding security facilities to the project was not that easy. It required the team members to make extensive research and write additional classes which increased the coding costs.

It was an important experience for us to communicate with stakeholders living abroad. Their suggestions and ideas concerning the design of a voting system helped us to broaden our visions and minds. Nevertheless, since we communicated them via e-mail, the lack of a personal encounter added difficulty to the requirements elicitation.

We have also learned to use the methodology of a spiral process. We wanted to have a working project from the beginning and add additional features to it one after the

other such that we could monitor our process and the state of the project more easily. We experienced the difficulties in changing requirements and/or design issues and reflecting these abstract changes to the documents and to the implemented code. The costs of testing and its importance was also a major lesson learned when doing this project. Trying to detect bugs and fixing them in a relatively large project (compared to the projects we have done for other courses before) can be really cumbersome and we spent more hours than we thought in testing this software. More detailed benefits of the spiral process are mentioned throughout the document.

Last but not least, we have seen the psychological advantage of taking small steps in completing the relatively more complicated software product. If we had the whole project to implement, life would be harder. Adding some functionality to some program that is working makes it seem as an easier goal. Testing would also include more complexity and hardiness when the whole software was to be tested at once instead of testing each additional part. To sum up, we have seen that the spiral process has psychological advantages over the waterfall process when the project size is small and the cost of building a prototype is not so high.

11. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

- GUI: Graphical User Interface, a type of user interface which allows people to interact with electronic devices
- UML: Unified Modeling Language, a standardized general-purpose modeling language in the field of software engineering
- SMS: Short Messaging Service, a communications protocol allowing the interchange of short text messages between mobile telephone devices.
- 3g: Third Generation of Mobile Phone Standards. 3G networks enable network operators to offer users a wider range of more advanced services while achieving greater network capacity
- Wi-Fi: Wireless Fidelity, a wireless technology. A Wi-Fi enabled device can connect to the internet when within range of a wireless network.

- Jpeg: Joint Photographic Experts Group, a commonly used method of compression for photographic images
- Quorum: the smallest number of votes allowed for the poll to count
- Horizontal structure: companies with little hierarchy where all employees have a say

12. REFERENCES

[1] FELDER, M., Distributed Decision in a Mobile Context, http://score.elet.polimi.it/projects/felder.pdf

[2] Jscape Secure iNet Factory, Trial Version, 2008, http://www.jscape.com/secureinetfactory/index.html

[3] JFreeChart library, 2005, http://www.jfree.org/jfreechart/index.html

[4] SOMMERVILLE, I., (2004). Software Engineering, Adison-Wesley, 7th edition.

[5] Prest, 2009, Department of Computer Engineering, Bogazici University, http://code.google.com/p/prest/

[6] NASA WVU IV & V Facility, Metrics Program. 2004. http://mdp.ivv.nasa.gov