

Universidade do Minho
Escola de Engenharia

Nelson Manuel Oliveira de Faria

AMPLIDIR - A Software System for Social Support to Decision

Tese de Mestrado
Mestrado Engenharia Electronica Industrial e Computadores

Trabalho efectuado sob a orientação do
Professor Doutor Paulo Garrido

Acknowledgements

I want to express my gratitude to Prof. Paulo Garrido for sharing all his passion for learning, the persistence during this work and his dedication to make this work possible.

I am in debt with my parents, which worked hard to give me the possibility to reach this objective.

Finally, I want to dedicate this dissertation to my wife and my daughter. Without their patience, support and love this dissertation would not have been possible.

Resumo

A exploração e aproveitamento do conhecimento do maior número possível de membros nos processos de decisão pode estimular substancialmente o sucesso das organizações. Através do alargamento e aprofundamento da participação dos membros é possível aceder a um maior leque de percepções, decisões alternativas e suas implicações.

Esta dissertação descreve os passos seguidos para definir os requisitos, especificação e implementação de um protótipo de Social Decision Support System (SDSS). A abordagem para a criação do protótipo foi a reutilização de um *bug tracker* em código aberto, pela semelhança com os requisitos seleccionados para a construção do protótipo.

O desenvolvimento do protótipo de SDSS utilizou o *bug tracker* Mantis e implicou a rescrita de parte deste. A arquitectura deste providenciou a autenticação de utilizadores, gestão de utilizadores, funcionalidades de email, tratamento de recepção de ficheiros e gestão de sessões.

As funcionalidades implementadas no protótipo SDSS incluiu a discussão de questões, perguntas/respostas e votações com distribuição de utilizadores por áreas.

O nível de desenvolvimento do protótipo permite testá-lo em organizações de forma a recolher informações para as funcionalidades implementadas.

Ao longo deste trabalho, a reutilização de software de código aberto mostrou ter vantagens para prototipagem rápida, porque resolve grande parte dos problemas iniciais de implementação.

Abstract

Exploring and exploiting the knowledge of as many members as possible in decision processes can foster substantially the success of an organization. Through widening and deepening members participation it is possible to access a wide range of perceptions, decision alternatives and their implications.

This dissertation describes the steps for a Social Decision Support System (SDSS) prototype requirements definition, design and implementation. The approach for creating the prototype was to reuse an existing open source bug tracker, because its functions are similar to the selected requirements for the prototype under construction.

The basic software development of the SDSS prototype proceeded as a rewriting of the open source Mantis bug tracker. The architecture of this one provided user authentication, user management, email functionalities, file upload handling and session management.

Features implemented in the SDSS prototype include issues discussion, questions/ answers and polls, with distribution of users among different areas.

The prototype was developed to the level where field testing in organizations can be conducted to collect feedback for the implemented features.

Along this work it showed up that reusing existing open source software for fast prototyping has advantages, as it solves most of the initial problems in project implementation.

Contents

1	Introduction	1
1.1	Decision processes in organizations	1
1.2	Objectives	3
1.3	Dissertation organization	5
2	Decision Support Systems	7
2.1	Decision Support Systems study	7
2.1.1	Evolution of Decision Support Systems	7
2.1.2	Types of Decision Support Systems	9
2.2	Decision Support Systems usage in organizations	10
2.3	Social Decision Support Systems	13
2.3.1	The potential of SDSS	15
2.4	Features for a generic SDSS	18
2.4.1	Objectives	19
2.4.2	System features	19
2.4.3	Opportunity for development of a Social Decision Support System	23
2.5	Usage examples for SDSSs	23
3	Free and Open Source Software and Bug-Trackers	25
3.1	Free and Open Source Software	25
3.2	Bug trackers - Possible approach for basis of implementation	26
3.3	Available bug trackers and features	27
3.3.1	Commercial systems	27
3.3.2	Open source systems	32
3.3.3	Bug trackers functionalities	34
3.4	Commercial versus Open Source bug trackers	36
3.5	Using a bug tracker as base for development	37
4	Prototype Design	39
4.1	Prototype objectives	39

4.2	Prototype features description	39
4.2.1	Users authentication	39
4.2.2	Bug to issues and project to area transformation . . .	40
4.2.3	Issue submission and issue management	40
4.2.4	Questions submission and answers collecting	41
4.2.5	Voting system	42
4.2.6	Issue “tagging”	43
4.3	Design proposal for a DSS - Amplidir	44
4.3.1	Assumptions and dependencies	44
4.3.2	Architectural strategies	44
4.3.3	System architecture	45
4.3.4	Questions and answers	46
4.3.5	Voting system	48
4.3.6	Issue tagging	49
4.3.7	Detailed system design	49
5	Prototype Implementation	51
5.1	Amplidir prototype implementation	51
5.1.1	Development environment	51
5.1.2	Test environment	53
5.2	PHP Language Overview	53
5.2.1	PHP Introduction	53
5.3	Implementation approach	55
5.4	Changes in the software design	57
5.5	Help material to users	57
5.6	Prototype Interface	58
5.6.1	Login page	58
5.6.2	Entry page	58
5.6.3	Main Menu	59
5.6.4	Issue List	60
5.6.5	Issue comments tagging	60
5.6.6	Issue fast feedback	61
5.6.7	Issue searching	61
5.6.8	Question list	62
5.6.9	Question detail	63
5.6.10	Question submission	64
5.6.11	Poll list	64
5.6.12	Poll detail, voting and results detail	65
5.6.13	Poll submission	66
5.7	Tests performed	67

6 Conclusion	69
6.1 Results	69
6.2 Advantages and disadvantages using the open source Mantis bug tracker	70
6.3 Future work	71
6.3.1 Field test of Amplidir	71
6.3.2 New features to add	71
A Questions API	73
A.1 Class QuestionData	73
A.2 Functions	73
A.3 Classes and functions to be developed in questionanswer_api.php	74
A.4 Functions	74
A.5 Questions database schema	75
B Polls API	79
B.1 Class PollData	79
B.2 Functions	79
B.3 Functions to be developed in poll_voting_api.php	80
B.4 Voting system database schema	80
C Issue tagging functions and database schema	85
C.1 System actions	85
C.2 Issue tagging database schema	85
D Detailed system design: action diagrams	87
D.1 Detailed system design: Questions action diagrams	87
D.2 Detailed system design: Polls action diagrams	95
E PHP Language Overview	101
E.1 PHP Introduction	101
E.2 PHP Variable types	103
E.3 PHP Operators	104
E.4 Assignment Operators	104
E.5 Bitwise Operators	104
E.6 Comparison Operators	105
E.7 Execution Operator	105
E.8 Incrementing/Decrementing Operators	105
E.9 Logical Operators	106
E.10 Array Operators	106
E.11 PHP Control structures	106

E.12 PHP functions	108
E.13 Function arguments	108
E.14 Function arguments by reference	108
E.15 PHP classes	109

List of Tables

2.1	Information gaps	16
3.1	Bug tracking commercial solutions	32
3.2	Bug Tracking open source solutions	34
4.1	User interface source code files description	47
4.2	Library source code files description	47
4.3	Voting system user interface source code files description . . .	49
4.4	Voting system library source code files description	49

List of Figures

2.1	Murray Turoff model	17
2.2	Super areas	18
2.3	Areas Interception	19
4.1	System overview	46
5.1	Mantis list	52
5.2	Amplidir issue list	52
5.3	Login page	58
5.4	Entry page	59
5.5	Main menu	60
5.6	Issue tagging	61
5.7	Fast feedback	61
5.8	Issue search	62
5.9	Issue search results	62
5.10	Question list	63
5.11	Question detail	63
5.12	Question submission	64
5.13	Poll list	64
5.14	Poll detail	65
5.15	Poll voting	65
5.16	Poll results detail	66
5.17	Poll submission	66
A.1	Questions Database schema	75
B.1	Voting database schema	81
D.1	Create Question	87
D.2	Create a Question (question submission)	88
D.3	View Question list	89
D.4	View Question	90

D.5 View Question: Answers list	91
D.6 Modify Question Status	92
D.7 Delete Question	93
D.8 Add Answer	94
D.9 Create a Poll	95
D.10 View Poll list	96
D.11 View Poll	97
D.12 Vote on Poll	98
D.13 Add comment to a Poll	99

Chapter 1

Introduction

1.1 Decision processes in organizations

Herbert Simon[Simon, 1976], defined a decision as a selection from a number of alternatives, directed toward an organizational goal or subgoal.

Accordingly, the selection of the alternative that results in the best set of all possible consequences requires three steps (*ibidem*):

- Identification and listing of all the alternatives;
- Determination of all the consequences resulting from each of the alternatives;
- Comparison of the accuracy and efficiency of the consequences resulting from each alternative.

These steps may be carried with varying degrees of breath and depth, implying associated costs of time and resources, from seconds of thinking of one person, to several months (or years) of many people involvement. Carrying the steps leading to a decision may be understood as a decision process.

In an organization many decisions must be taken, so one may say that at any given instant a set of (inter-related) decision processes is taking place. Success or insuccess of an organization depends critically on its decision processes delivering decisions adequate, in time, number and quality, to the fulfilling of the organizations goals.

Awareness of the decision processes occurring in an organization takes several forms, for example, through the stating of a formal model as an organizational chart. Yet, decision processes inside organizations do not occur only accordingly to its formal model, but immersed in the organization social reality.

The social network in the organization influences problems formulation, solutions created and even the decision consequences perception.

Organization members interact in a social complex social system in many ways. Within an organization, all the people, their relationships to each other and to the outside world constitute the social system. The behavior of one element can affect directly or indirectly others' behavior.

The social component is inherent to every decision process. This component can be observed from different viewpoints:

- From inside the organization, as the members' connections, influences, hierarchy, leadership, sense of group behavior;
- From moral values (religion, preconceptions, political activism), external to the organization, yet creating strong behaviors in the members that run for that rules;
- From society laws, that compel the organization not to adopt illegal decisions that constrain its behavior so and so. Sometimes these laws create inside organizations lobbies that follow the rules or try to escape them.

Not accepting that this component is part of any decision process and not involving all the members into the process can create some negative, yet natural, social behaviors, as groups inside the organization looking for protection or power. This can produce confrontation instead of common effort.

Assuming that the inherent social component in organizations is essential to organizational performance may lead into getting:

- Members not fearing punishment as a result of thinking differently from the majority of the other members;
- Increased organizational perception about the reality, problems and decision effects;
- Increased true competence recognition inside the organization, based not on politics and status, but on the efforts to reach common goals and encompassing all organization members (global organization recognition);
- Access to information the organization does not know it exists;
- Capture of improbable and rare ideas. These improbable and rare, deviating from average, ideas create the greatest opportunities and

generate the major changes. Nicholas Taleb [Taleb, 2007] called them Black Swans: "Black Swan logic makes what you don't know far more relevant than what you do know. Consider that many Black Swans can be caused and exacerbated by their being unexpected.". The black swan theory refers to a large-impact, hard-to-predict, and rare event beyond the realm of normal expectations.

Scott Peck [Peck, 1987], created a model with different social stages for large scale groups. The stages by level of involvement are:

- Pseudo-community: This is a stage where the members pretend to have same objectives, and cover up their differences, by acting as if the differences do not exist. Pseudo community can never directly lead to community, and it is the job of the person guiding the community building processes to shorten this period as much as possible;
- Chaos: When pseudo community fails to work, the members start falling upon each other, giving vent to their mutual disagreements and differences. This is a period of chaos. It is a time when the people in the group realize that differences cannot simply be ignored. Chaos looks counterproductive but it is the first genuine step towards community building;
- Emptiness: After chaos comes emptiness. At this stage, the people learn to empty themselves of those ego related factors that are preventing their entry into community. Emptiness is a tough step because it involves the death of a part of the individual. However, Scott Peck argues, this death paves the way for the birth of a new creature, the Community;
- True community: Having worked through emptiness, the people in community are in complete empathy with one another. There is a great level of tacit understanding. People are able to relate to each other's feelings. Discussions, even when heated, never get sour, and motives are not questioned.

1.2 Objectives

The main objective of this work was to create a software platform prototype to help organizations to harvest many of the potential benefits of involving as much as possible all their members in decision processes.

The software prototype is neutral concerning to questions of authority, about who takes decisions and how to take decisions. So, it is flexible

enough to accommodate varying degrees of participation according to the organization "decisional" culture, values and practices.

This special type of Decision Support System (DSS), namely Social Decision Support System (SDSS), has been named Amplidir, standing for amplify directions (decisions).

Amplidir configures a set of tools for social support which can be used with different degrees of involvement in the organization path to a true community or, simply, to make a more efficient organization. The intended consequences of using it can be related to the enhancement of the characteristics of a true community, as defined by Scott Peck (*ibidem*);

- Inclusivity, commitment and consensus: Members accept and embrace each other, celebrating their individuality and transcending their differences. They commit themselves to the effort and the people involved. They make decisions and reconcile their differences through consensus.

Amplidir: a platform for all to be included in decision process, to enable all to share and commit to decisions.

- Realism: Members bring together multiple perspectives to understand the whole context of the situation. Decisions are more well-rounded and humble, rather than one-sided and arrogant;

Amplidir: a platform enabling the different perspectives to be explained and to obtain feedback from other members.

- A group of all leaders: Members harness the flow of leadership to make decisions and set a course of action. It is the spirit of community itself that leads and not any single individual;

Amplidir: the decision will be known by everyone, even before its communication. The discussion serves as introduction.

- A spirit: The true spirit of community is the spirit of peace, love, wisdom and power. Members may view the source of this spirit as an outgrowth of the collective self or as the manifestation of a Higher Will.

Amplidir: Is a tool to help members to take decisions and to progress together as a community.

To prepare the design and system implementation, the work was broken in the following steps:

- Study of different types of DSS;

- Study of the evolution of the DSS;
- Study of current work in the SDSS field (papers and software platforms);
- Create list of features for a generic SDSS;
- Study the possibility to use an open source system (bug tracker);
- Study of the existing open source “bug trackers” systems;
- Choice of an open source “bug tracker” system software to be the base for implementation;
- Prototype design and selection of the features to be included as proof of concept;
- Prototype implementation.

The result of this work is a prototype with source code ready for download and a demo instance on line for user testing. The software platform used the bug tracker “Mantis” as development base and has the following features:

- Issues submission and discussion notes with different discussion types;
- User tagging of discussion notes;
- Questions and answers module;
- Polls and voting module;
- Simplified issues/notes/questions search.

1.3 Dissertation organization

The dissertation has the following organization. In Chapter 2, we can find the study of Decision Support Systems (DSSs): different DSS types and evolution, performed as start for the project. In this chapter, we can find the definition and features for a generic “Social Decision Support System”.

Bug trackers offer a good base for development start, because they already address one important aspect of a DSS: the question addressing to other organization elements and the reply/information collecting to answer questions. Chapter 3 is a study for the existent open source bug trackers.

After the study and with the bug tracker selected, the next project phase was to select the features for the prototype and design the system. The chapter 4 presents the Amplidir prototype design.

In Chapter 5, prototype implementation details are explained, with the implementation approach and the changes needed to the initial design. The tests description performed in the system are in this chapter.

Chapter 6 describes the results, conclusions and future work.

Chapter 2

Decision Support Systems

2.1 Decision Support Systems study

2.1.1 Evolution of Decision Support Systems

The origin of DSSs goes back to the 1960 decade, when researchers created computerized models to assist in decision making and planning [Power, 2007].

Licklider [Licklider, 1992] published his ideas about the future role of multi-access interactive computing in a paper titled “Man-Computer Symbiosis”.

By 1964, IBM introduced powerful mainframe computers that made cost-effective and practical to build Management Information Systems (MIS) for large companies. These systems provided periodic reports based on accounting and transaction systems.

Soon after, Scott Morton [Morton and Stephens, 1968] and colleagues published a number of decision support articles. Ferguson and Jones [Ferguson and Jones, 1969] discussed a computer aided decision system in the Management Science journal.

Gerrity [Gerrity, 1971] wrote an article titled “The Design of Man-Machine Decision Systems: An Application to Portfolio Management”. His system was designed to support investment managers in their clients’ stock portfolio administration.

Scott Morton [Morton, 1971] published a ground breaking book *Management Decision Systems: Computer- Based Support for Decision Making*. This was the first time the term “Decision Support System” was introduced.

Alter [Alter, 1975] finished his PhD. dissertation named “A Study of Computer Aided Decision Making in Organizations”, contributing to expansion of the DSS thinking.

Hackathorn and Keen [Hackathorn and Keen, 1981] split DSS in three

distinct categories: Personal DSS, Group DSS and Organizational DSS.

Following this, other researches published studies with relevance, for example: Ralph Sprague and Eric Carlson's published in 1982 the book *Building Effective Decision Support Systems*. They defined a DSSs as a "class of information system that draws on transaction processing systems and interacts with the other parts of the overall information system to support the decision-making activities of managers and other knowledge workers in organizations" (p. 9).

The first International Conference on Decision Support Systems was held in Atlanta, Georgia in 1981. Academic conferences provided forums for ideas sharing, theory discussions and information exchange.

In the early 1980s, academic researchers developed a new category of software to support group decision-making called Group Decision Support Systems abbreviated GDSSs ([Gray, 1981]; [Huber, 1982] and [Turoff and Hiltz, 1982]). Mindsight from Execucom Systems, Group-Systems developed at the University of Arizona and the SAMM system developed by University of Minnesota researchers were early GDSSs.

In 1990 Red Brick Systems introduced Red Brick Warehouse, a database management system specifically targeted to data warehousing. This database, designed to support decision making in an organization, collects data from the production databases so that queries can be performed without disturbing the performance or the stability of production systems.

This new concept is now known as data-driven DSSs. As an interesting note, in 1997 the Wal-Mart and Teradata created the world's largest production data warehouse at the date, with 24 Terabytes.

Heylighen [Heylighen, 1999], published the paper "Collective intelligence and its implementation on the web: Algorithms to develop a collective mental map", describing several algorithms for collective problem solving through mental mappings. Examples are:

- Averaging preferences: Instead of simple voting, members can put an amount of vote by preference. For example: 0.5 for option A, 0.3 for option B and 0.2 for C. This is to some degree similar to the functioning of ant colonies, where the pheromone trail left by a particular ant can be seen as that ant "vote" in the discussion of where best to find food ;
- Feedback: Achieved by discussion, proposing solutions and arguments to support it. These arguments may convince others that an option is really the best, or incite them to produce counter-arguments. In the best case, these arguments and counter-arguments will illuminate the

broader implications of the different options, or even suggest a new option that combines the best aspects of the previous options;

- **Division of labor:** The work is split among different members and concentrated in a manager group or individual that will synthesize it in a broader perspective.

Murray Turoff [Turoff et al., 2002] proposed a social decision support system with a process model to achieve collective decision by dynamic voting (the vote can be changed at anytime of the discussion process). One of the most interesting aspects of this type of system is that the voting process must be continuous and it must be of such a nature as to help filter and organize the resulting material.

Rodriguez and Steinbock [Rodriguez and Steinbock, 2004] introduced a new concept in social trust networks. Instead of simple trust links like element A trusts element B, he proposed links of levels of trust, such as: element A trusts completely in element B, however element B trusts 0.2 in element A and 0.8 in element C.

In 2007, the paper “Smartocracy: Social Networks for Collective Decision Making” [Rodriguez et al., 2007] described a software system to support large groups to take collective decisions. The collective decision is formed by direct and indirect proxy voting, where an element’s vote is counted by itself and for the non voters that trust him.

2.1.2 Types of Decision Support Systems

Daniel Power [Power, 2002] proposes a classification for DSSs:

- **Model-driven DSS**

This type of system uses data and parameters defined and provided by decision makers to aid analyzing a situation, but they are not usually data intensive. Dicodess ¹ is an example of an open source model-driven DSS generator.

- **Communication-driven DSS**

Most communications-driven DSSs target internal teams, including partners. Their purpose can be to help conduct a meeting or users’ collaboration through data sharing. The most common technology used to deploy the DSS is a web or client server. Examples: chats and instant messaging software, on line collaboration and net-meeting systems.

¹<http://dicodess.sourceforge.net>

- **Data-driven DSS**

Most data-driven DSSs target managers, staff and product/service suppliers. They query a database or data warehouse to seek specific answers for specific purposes. They are available via a mainframe system, client/server link, or via the web. Examples: computer-based databases that have a query system to check (including the incorporation of data to add value to existing databases).

- **Document-driven DSS**

Document-driven DSSs are more common, targeted at a broad base of user groups. The purpose of such a DSS is to search web pages and find documents on a specific set of keywords or search terms. The usual technology used to set up such DSSs is via the web or a client/server system. Actually, this technology is in personal computers using the software: Google Desktop Search or Microsoft Windows Search.

- **Knowledge-driven DSS**

Knowledge-driven DSSs can suggest or recommend actions to managers. These DSSs are person-computer systems with specialized problem-solving expertise. The “expertise” consists of knowledge about a particular domain, understanding of problems within that domain, and “skill” at solving some of these problems.

2.2 Decision Support Systems usage in organizations

In the 80's, organizations' managers put many efforts building and using DSSs. Unfortunately, the output was not satisfactory and most companies just created systems to address specific topics for the business organization. Every business has some particularities and the major advances were in very powerful database systems.

Database systems and datawarehouse technology widespread the DSSs systems usage.

There are several systems used for decision support, using database mining or supporting group decisions in everyday activities, for example:

- **Chordiant Customer Experience²**

System to help to take the best decisions on customers contacts. The

²<http://www.chordiant.com/solutions/>

decisions must reflect the business strategy, the interests of the customer, their value and risk to the business. This software ensures such approach.

The system has multiple views for different company activities. The major company using this software is Orange³.

- **FacilitatePro**⁴

System classified as group DSS to help group problem solving and decision making in on-line meetings.

The system has focus on the tasks: brainstorming, categorizing, prioritization, voting, action planning, surveying and documenting.

- **Microsoft Office SharePoint Server**⁵

System that helps users to access centralized information about the business, projects and activities.

The system has the possibility for users to create their own content and share among the partners.

- **Vanguard System**⁶

System that covers all the organization activities, such as:

- Planning & Analysis

- Collaboration in projects and activities, analysis and simulation of alternatives.

- Knowledge Automation

- For call center scripting, customer self-service troubleshooting, product configuration and sales scripting.

- On line surveys

- For customer satisfaction and employee surveys.

- **IBM Cognos 8 Business Intelligence**⁷

Reporting system and dashboard to monitor, control and take best data based decisions.

One of the main features is the ability to let users define and share reports over the existing data sources (self-service reporting).

The U.S. Department of Defense uses this software to control and plan the financial budget.

³http://www.francetelecom.com/en_EN/

⁴<http://www.facilitate.com>

⁵<http://office.microsoft.com/en-us/sharepointserver>

⁶<http://www.vanguardsw.com>

⁷<http://www.cognos.com/>

- **Business Objects** ⁸

The software helps organizations gain better insight into their business, improving decision-making and enterprise performance.

The users can define and share reports over existing data.

Other application field is in medical care, with the following systems available:

- **TheraDoc - Expert System Platform** ⁹

The TheraDoc Expert System Platform provides active surveillance and real-time tools that enhance decision-making. The software has some useful components:

- ‘Rounds Assistant to assist in patient data collection;
- Intervention Assistant to automate the medical staff through pre-defined workflows;
- EZ Alerts Assistant to create alerts based on system collected data.

- **VisualDx** ¹⁰

Designed for public health, emergency, and primary care clinicians, the VisualDx clinical decision support software system guides diagnosis, treatment, and management of diseases including emerging infectious diseases (EIDs) and conditions potentially caused by bioterrorism, chemical warfare, or radiation release.

VisualDx helps clinicians rapidly build a custom, visual differential diagnosis based on their patient’s unique findings (e.g., symptoms, signs, lesion type, body location, radiological findings, medical history, travel, etc.) and quickly drill down to medical images and useful, actionable clinical information on each possible agent condition.

- **DXplain** ¹¹

DxPlain was developed by the Laboratory of Computer Science at the Massachusetts General Hospital. The software has a database for more than 2200 different diseases.

The software has an interactive workflow for signs and symptoms data collecting, allowing to select the potential diseases on these symptoms and suggesting which further clinical information would be useful to collect.

⁸<http://www.businessobjects.com/>

⁹<http://www.theradoc.com>

¹⁰<http://www.logicalimages.com>

¹¹<http://lcs.mgh.harvard.edu/projects/dxplain.html>

2.3 Social Decision Support Systems

Collective decision-making is central to collective action. The *overload problem* occurs when a collective does not have the information-processing infrastructure to support the active participation of all its constituent members in all relevant decision-making processes [Fischer, 1999] [Rodriguez, 2005].

The social component is intrinsic to the decision process, even if is not explicitly assumed. One can define as social component, the connections between people, direct or indirect, for example: hierarchy links, confidence relations, recognized expertise, support of group members, influence opinion makers, social meta decisions (laws, culture, moral, religion).

Recognizing the social interaction as a vital component of a decision process will exploit the advantages of having all members involved in the decision process.

The main advantages of this involvement are:

- Producing of more or alternative options or solutions to existing problems;
- More perception about the problem;
- Similar or parallel problems could enhance decisions or benefit from a wide involvement;
- Perception about problems that could be created from the proposed solutions.

A SDSS has as main objective to support active participation of all members in the decision process.

The system will help the members to actively participate in problems discussions, share options, participate in polls or agree with solutions. The system must have tools (anonymity, explicitly answer request) to prevent some group behaviors that in certain situations lead to bad decisions.

Known group critical factors to group decisions are:

- **Majority consensus and silence**

Studies show that members prefer to agree with the majority group opinion rather than to express their own, even when they know the group is wrong.

In the words of Paulus and Nijstad [Paulus and Nijstad, 2003] “Some researchers (Summerfield, 1990) estimate that at least 7 out 10 people in American business remain silent when their opinions are at odds with their superiors. Even when they know better, they permit their

boss to make mistakes.”

People adopt this type of position (silence), because they fear punishment, which in fact happens in non-open environments. This has been termed “organizational silence” [Morrison and Milliken, 2000].

- **Polarization**

When the group has very similar perception about a problem, decisions could be very risky.

Other situation occurs when the group members are over confident about the solution and aligned inside the group.

For example, if database creator team makes a reunion to analyze the database performance, creators tend to select factors external to team responsibility to blame for database performance, as: poor application programming from programmers using the database.

A person new to the team could challenge the polarized group consensus.

- **Different participants background**

In a way very similar to majority consensus, people often do not expose their opinion because the other elements are more experienced, with more education or are from a different hierarchy in the organizations. Most times the people fear criticism because they do not expose their opinions using the right terms or language or they fear their ideas are challenging the other elements having better background.

In an open collaborative group, this different participants background is healthy and useful for a better collaborative decision.

- **Ignorance judgment and psychology cost**

Some experiences by Gerstberger and Allen [Gerstberger and Allen, 1968] proved that in very technical job positions people do not search for the information among their colleagues because they fear the ignorance judgment. They choose not to go by the channel of the highest quality for technical information, but rather to go to the channel of highest accessibility (lowest psychological cost).

The use of a software platform by itself is not enough to promote a collaborative environment. According with Alan MacCormack [MacCormack, 2007], the four pillars of collaborative capability are:

- People
- Process

- Platform
- Program

The organization managers must understand the concept of collaborative participation and promote an open collaborative environment.

A process must be in place and used by the team. The process could consist in reporting rules and voting participation for the organization elements.

2.3.1 The potential of SDSS

The platform to help an organization to construct the confidence to share knowledge and help the managing board to take effective decisions must enable the members to express themselves without fearing judgments.

This means the system must provide some functionality to protect the members' identity enabling free opinion making.

Some elements do not have the confidence to express themselves in text and for them the voting systems with options are best.

Despite the human nature to help and share, the organization competitive environment does not foster such natural behaviors. To improve collaboration the organization could have a Chief Collaboration Officer with the task of improving the performance of all collaboration efforts.

The first attempt to use these social links was made by Murray Turoff [Turoff et al., 2002] with a new approach to solve problems, using a human-collective problem-solving algorithm. He proposed a process model for a SDSS, shown in Figure 2.1. Collective problem-solving, is achieved through collecting solutions and voting.

Existing DSSs are oriented for information management and need some type of aggregation or data mining, due to the data quantity available. SDSSs are different because the management layer could access information it does not know to exist inside the organization.

On the other way, the SDSSs are oriented to involve all the members in the decision process, in a way that collective knowledge could emerge.

Table 2.1 shows the possible information gaps, and where SDSSs could help to take better decisions.

With the extension to the system proposed by Murray Turoff to accept information from the organization, the system can capture information that the organization was not aware and that can have great potential.

	Have	Do not have
Know	Information you know you have	Information you know you do not have
Do not know	Information you do not know you have	Information you do not know you do not have

Table 2.1: Information gaps

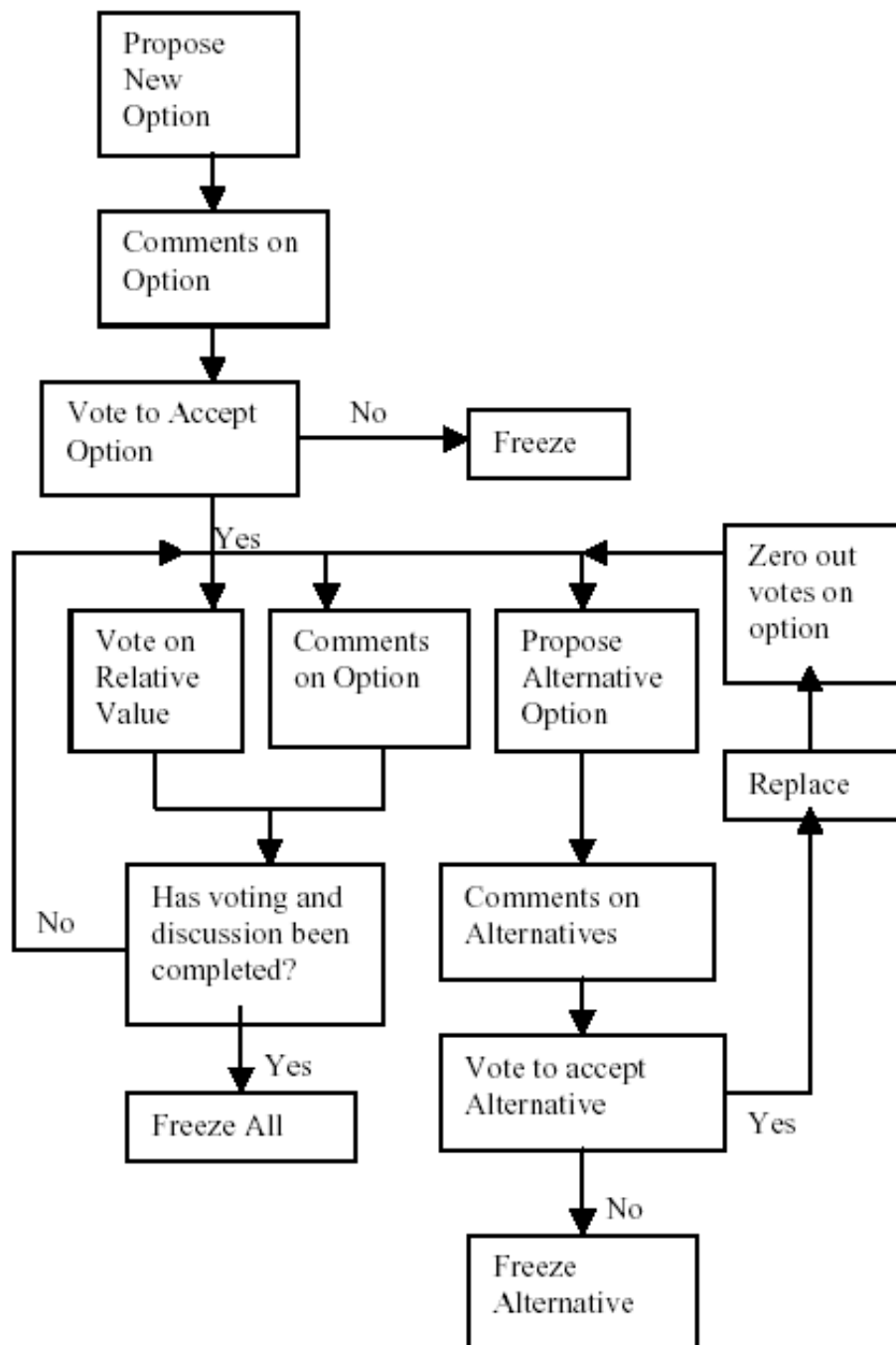


Figure 2.1: Murray Turoff model

2.4 Features for a generic SDSS

A SDSS must be capable to collect ideas, proposals, issues, concerns, information with or without request and provide ways to discuss and produce decisions.

It must also provide a mechanism to:

Formulate the problem → collect options for possible solutions → voting on options → classify the solutions.

Some cases could require a “brain storm model”:

Formulate a problem → collect all opinions → share all opinions at same time → optional voting.

Every organization has members (people) that work together (groups), and these groups interact or intercept in interests or objectives. To capture the organization static and dynamic structure, we will use the term “areas” to organize the members and the decisions. Static structure refers to the organization hierarchy and dynamic structure refers to the organization projects or initiatives. See Figure 2.2.

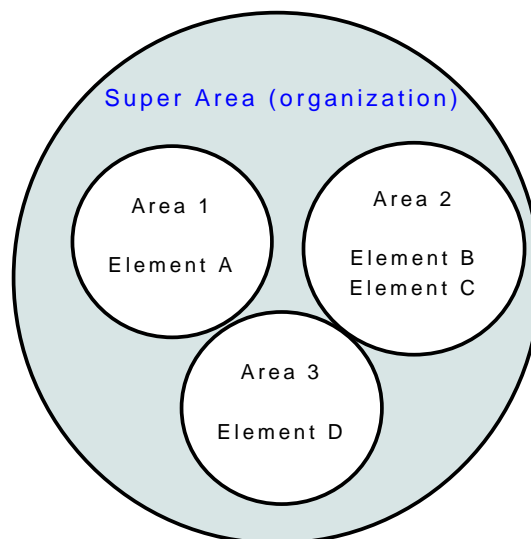


Figure 2.2: Super areas

By default, an area should have at least one element. One element could belong to several areas. See Figure 2.3.

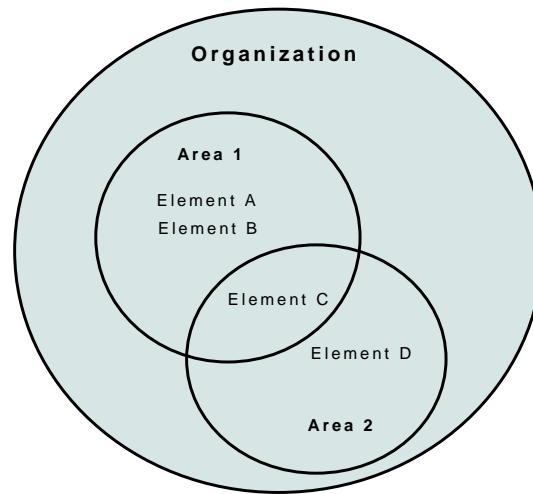


Figure 2.3: Areas Interception

2.4.1 Objectives

The system should be able to organize the members into areas and manage the problems in these areas. The system should also provide a way to enable communication and knowledge sharing and give members the possibility of self expressing without fear of criticism.

2.4.2 System features

For simplicity, we will consider *issue* any generic information introduced in the system deemed capable of generating a decision; it could be information, concerns, reports, etc. The requirements of a SDSS we will consider are:

- **Users and areas organization**

The system database must contain user information and the links about the status in the organization: Who belongs to each area.

The database must contain the access rights of users to the provided services: approvals, issues submissions, search.

- **Issues / knowledge database**

The database must be able to store the issues from users inside areas.

The database must store documents/files related with the issue.

The database must have the possibility to link issues/information.

The database must have the possibility to add comments to existing issues.

- **Voting**

The system must enable a manager to propose a voting with pre-

selected options and collect the voting results. It must also be able to be configured for the following situations:

- Do not show any results;
- Show results on line during the process with or without details;
- Show results when voting is finished;
- At any time, the system must show the percentage of participation and time remaining for the voting to end.

- **Complex Voting mechanisms**

- Direct proxy voting
This is the type of voting when the user selects a user to vote by him, if he or she does not vote.
- Indirect voting (trust relations)
The user vote has as much importance as more people select him as trusted. This is indicated for communities where the voting percentage is not considerable.
- Round voting
Voting by rounds. Each round eliminate half of proposals, until the final solution is accepted by the majority. This schema is vital when it is important to get majority acceptance and there are several options.
- Expert representation
Instead of solutions, votes choose organization members by expertise in topics. The experts will discuss and select the best option.

- **Questions / Answers**

The system must allow members to request help for a question. The participants could answer and optionally attach files to answers. The system must be capable of creating a simple voting system from the questions / answers process, and the manager must choose the options from those available answers. The system must have two different approaches on the identification of the question originator:

- Identify user;
- Do not show user identity.

The system must have the following modes for answers:

- Forum mode: show all answers as soon as they are available;
- Brainstorm mode: show all answers only when the process is finished;
- Open brainstorm mode: answers are shown, but without identification. Identification is showed at the end of the process;
- Inquire mode: answers are never shown.

- **Search capability**

The system must be capable to perform search through all the fields in the database.

The system must be able to save the performed searches.

It must be possible to create notifications from a performed search.

It must be possible to search the issues by tagging rank (see tagging section).

- **Decision flow approval**

The system must be capable of defining a decision flow approval per area of knowledge.

The system must warn the managers that they have issues to approve.

The states of the decision flow must have at least the following attributes:

- Name
- Approval users
- Current state
- Next state

- **Classification component**

Issues should be classified in categories according to knowledge areas involved. The system must have a component to propose a category to the user when he does not select one. This is very important to reduce the number of unclassified issues.

It must be possible to reclassify a submitted issue. The issue must be transferred to the first state of the selected category.

- **Tagging system**

One functionality of a social network present the web is the “tagging” feature available on the site digg.com¹². At the site, users vote on the web pages by importance (no pre-defined rules for importance or social

¹²<http://www.digg.com>

interactions: environment will define a natural method) and classify the web page by categories.

In a similar way, the users must be able to select issues, vote on them, and optionally attribute them to a predefined category. This is a generic capability to be used if the social network wants to. It could be used for example for alerts in issues that the community considers deserving some attention.

- **Email processing system**

The system must be capable to access an email box (POP3), and process new messages as new issues.

The system must be able to send messages to users about approval requests and to process the reply with the selected option.

The system must be able to send an email to users requiring approval needed and process the reply with selected option.

The system must be capable to send an email requiring a vote in an open voting process and process the reply as a vote.

All this types of automated email interaction must be configurable by the user.

The system must have a mechanism to ensure that the message is a reply from a server sent message.

- **Web GUI interface**

The web interface must have an authentication schema (optional integration with Active Directory).

The web interface must have access to all features defined above.

The interface must allow to configure and administer users and groups.

- **MyDesk system**

The MyDesk is like a personal view over the SDSS. It must have:

- My approvals (issues needing approval);
- My issues (submitted issues);
- My searches;
- My management requests (voting or questions from management);
- Top-tagged issues;

2.4.3 Opportunity for development of a Social Decision Support System

Nowadays if a company wants to use a SDSS system, the only option is to build a system from scratch, or use multiple tools that implement part of the desired integrated system, like voting systems. Searching along all the different systems is problematic and to correlate different types of information is difficult for a generic system.

Bug tracker systems have some similarities with a SDSS, bug tracking being a very close concept to decision making flow:

- Members can submit the bugs;
- Other members can add more information to the discussion or even propose solutions;
- The bug can be closed (have states).

There are missing features to help the group to take decisions as a group, such as voting or solutions' classification.

The usage of an open source bug tracker system as base for development is a good solution for re-use efforts in prototype implementation.

There is a window of opportunity to build an open source system that would be used widely and grow in features when more interested developers join the project, since there is no available open source solution.

The open source strategy is best for this type of application because each company has their own special requirements and companies with enough resources could help in the effort to create a better SDSS for everyone.

2.5 Usage examples for Social Decision Support Systems

Every organization can use a SDSS to involve all members in decision processes. There are organizations where this type of system seems to fit better, for example:

- Organizations where the expertise is not focused in the information available but in the information analysis, *i.e.* Hospitals.
Doctors could use the platform to share the problems they face and collect others' opinions to take the best decision.
A system providing voting and tagging capability could help the problem owner to understand which opinions other doctors most support;

- Organizations where the persons are geographically distributed and need each others help, *i.e.* scientific research;
- Organizations addressing distributed problems over different locations: organizations with production sites in different places that need to share knowledge about process methods and logistics between sites;
- Organizations addressing very complex and distributed problems: national or international police. In this case, information from other places could help to solve local cases, as hidden connections could be found.

Chapter 3

Free and Open Source Software and Bug-Trackers

3.1 Free and Open Source Software

According to Wikipedia [Wikipedia, 2008]: “Free and open source software, also F/OSS, FOSS, or FLOSS (for *Free/Libre/Open Source Software*) is software which is liberally licensed¹ to grant the right of users to study, change, and improve its design through the availability of its source code². This approach has gained both momentum and acceptance as the potential benefits have been increasingly recognized by both individuals and corporate players.”

The distinction between free and open source hinges on intentions and values behind the two definitions [Wikipedia, 2008]:

“F/OSS’ is an inclusive term generally synonymous with both free software and open source software which describe similar development models, but with differing cultures and philosophies. ‘Free software’ focuses on the philosophical freedoms it gives to users and ‘open source’ focuses on the perceived strengths of its peer-to-peer development model. Many people relate to both aspects and so ‘F/OSS’ is a term that can be used without particular bias towards either camp.”

On the above one can contend that the “freedoms it gives to users” are much more than “philosophical”.

Different intentions and values have not prevented FOSS from becoming a major transformational force of today’s world, including becoming a growing area of investment and revenue for the Information Technology business [Iansiti and Richards, 2006].

¹http://en.wikipedia.org/wiki/Software_licence

²http://en.wikipedia.org/wiki/Source_code

Software used world wide which is free and open source:

- GCC: a C cross compiler available to almost all systems
- Apache: most used web server in the internet
- Firefox: popular web browser
- MySQL: a database server used in most systems
- Linux: open source operating system
- OpenOffice: a office suite with word processor and spreadsheet
- CVS: source code control system
- Gimp: image manipulation
- Inkscape: vectorial image manipulation
- Java: language compiler and runtime
- Python: script language processor
- PHP: web language
- Sendmail: the project that supported the email in the internet

All these projects impacted the world as we know it today.

3.2 Bug trackers - Possible approach for basis of implementation

Bug/issue tracking systems are software programs to enable the submission, evaluation and follow up of issues.

In order to prioritize issues these systems usually have some type of issue classification.

A resolution workflow to approve and resolve issues is also common. At any time, it is possible to know who has the problem ownership, resolution history and all the documentation related with the communication between the reporter and the development team.

This model has mainly two different applications:

- Issue/ticket tracking systems: systems used in operations support (software applications, operating systems support);
- Bug trackers: systems used in software development and tests.

Bug trackers implement part of the communication layer needed by an SDSS. If the concept “project” is converted to “area” and “bug” to “issue” or “question”, reusing this type of software as the departure point for an SDSS is feasible.

Big software projects are using bug trackers software for social communication and interaction among developers and users.

Almost all bug trackers use a web interface as user interface to collect the communication between users and developers about problems found in the software. The “bottom to top” communication scheme has the property one desires for a SDSS: raise issues to be decided upon.

The web-based interface means low maintenance costs in the infrastructure with less distributed applications.

Open source bug trackers systems appear in most cases to support big open source projects such as Mozilla³, MySQL⁴ or GNU⁵.

3.3 Available bug trackers and features

3.3.1 Commercial systems

Table 3.1 shows several available commercial solutions for bug tracking in software development.

System	Architecture	Features
AgileEdge	Web based Developed in JAVA Database: Oracle 8.0 or superior MySQL 2.4 or superior Sql Server 2000	Email notifications Possibility to attach files to issues Changes history Easy to use interface

³<http://www.mozilla.org/>

⁴<http://www.mysql.com/>

⁵<http://gnu.org>

Arctic PHP Bug Tracker	Web based MYSQL database Implemented in PHP	Filter creations Email notification on issue state changes Import of other “Bug Tracker” databases Multiple languages by user configuration
Axosoft OnTime	Developed in C# Microsoft SQL Server database	Possibility to follow defects requests or tasks independently or together Possibility to change predefined workflow LDAP authentication support Predefined reports and new report creation
BugAware	Web based Implemented in ASP MS SQL Server database Runs on Windows Server	Definition of defects and tasks Email notification on issue state changes Multiple languages
BugZero	Web based Implemented in J2EE/JAVA Supports SQL92 standard	Email processing as new requests Possibility to change predefined workflow Automatic or manual assign Predefined reports LDAP authentication support
Clarity	Web based Implemented in J2EE/JAVA Proprietary filesystem instead of a database server	Definition of defects and tasks Email processing as new requests

FIT — BugTrack	Web based Works with or without database	Email notification on issue state changes Possibility to change predefined workflow
FrogBugz	Web based Implemented in ASP e PHP databases: SQL Server or MySQL	LDAP authentication support Different languages support by user Email notifications subscription on issue state changes Discussion groups
Geminai	Web based Implemented in C Database: SQL Server	Possibility to change predefined workflow Email notifications subscription on issue state changes Possibility to attach files to issues
Isósceles Inter-cepta	Web based Implemented in .NET Run on Windows Server	Possible to define workgroups by project Workgroups graphical representation Possibility to attach files to issues Email notifications subscription on issue state changes
Nereidas Tracker	Web based Databases: Microsoft Access, Microsoft SQL Server or Oracle	Email processing as new requests Email notifications subscription on issue state changes Multiple time zones supported Multiple projects and groups Multiple languages configurable by user
Rational Clear-Quest(IBM)	Web based Works on multiple platforms	LDAP authentication support Workflow definition Email notification on issue state changes

Serena Teatradas	Web based Run on servers: Windows Red Hat Linux AS 4.0 Solaris 9 Solaris 10 Databases: Microsoft SQL Server 2000/2005 IBM DB2 Universal Database v8.2 Oracle v9.2.0.7 or 10g Sybase Adaptive Server Enterprise version 12.5.4	Microsoft Access 2002 Multiple projects and groups The solution is part of the ITIL process Mobile platforms support
TestTrack Pro	Web based Databases: TestTrack native SQL Server 2000 Enterprise Oracle 9i, 10g MySQL 5.0 ODBC-compatible	LDAP authentication support Email notifications subscription on issue state changes Possibility to change predefined workflow
Track+	Web based Databases: SQL Server Oracle	Multiple projects and groups Email notifications Possibility to change predefined workflow Possibility to attach files to issues Email processing as new requests Multiple languages configurable by user

TrackStudio	Web based Operating Systems: Microsoft Windows NT/2000/2003/XP Linux Sun Solaris Hewlett Packard HP-UX IBM AIX FreeBSD Databases: ORACLE 8i, 9i, 10g IBM DB2 8.2 MS SQL Server 2000 SP3, 2005 Firebird 2.0 PostgreSQL 8.2 HSQLDB 1.8.0 MySQL 4.1	Multiple projects and sub projects Possibility to change predefined workflow Possibility to create custom fields by project and workflow Email Notifications Automatic email processing for: submitting, changing priorities Web based reports Reports exporting LDAP, Active Directory and NTLM support
Unfuddle	Web based MYSQL database	Web based reports Email notifications
VisionProject	Web based Implemented in JSP/Java Databases: SQL Server or MySQL	Supports issues, tasks and surveys Email notifications Automatic email processing Possibility to attach files to issues Possibility to change predefined workflow

Woodpecker Issue Tracker	Web based MySQL database Implemented in PHP	Predefined workflow (Recorded, In-Process, Repaired, Testing, Tested, Complete) Possibility to change predefined workflow No limits on projects or users Configurable permissions on user level Possibility to attach files to issues Email notification on issue state changes
yKAP	Web based Microsoft Windows Server Database:MS SQLServer	Possibility to change predefined workflow Email notifications

Table 3.1: Bug tracking commercial solutions

3.3.2 Open source systems

In almost all fields where commercial systems exist, the open source movement aggregates programmers that build together systems in open source. These programmers build open source systems to compete with proprietary systems or simply to program in areas they do not have other way to work on.

Table 3.2 shows available open source systems addressing bug tracking in software development.

System	Language Database	Features
Bug-A-Boo CGI	None	Fixed workflow
Bugzilla Perl	MySql	Speed database optimization Advanced search Email integration: notifications and processing Intuitive permission system

BugTracker.NET	.NET SQL Server	Workflow redefinition Email notifications Unicode support
Eventum Issue	PHP MySQL	Multiple projects Email notifications Email processing to create new issues Issue attachments support
Jtrack Java	Oracle MSSQLServer	Configurable workflow by project Email notifications Issue attachments support LDAP and Active Directory support Possible usage without database Multiple languages
Mantis	PHP MySQL	Multiple projects Subprojects and categories support Personal view Predefined reports Email notifications Issue attachments support RSS support LDAP and Active Directory support Webservice (SOAP) interface Mobile devices support
phpBugTracker	PHP MySQL	Multiple projects Permissions by project HTML templates for user interface customization Issue attachments support

RT: Request Tracker	Perl MySQL, PostgreSQL or Oracle	Multiple languages History and tendencies
Scarab	Java MySQL, Oracle	High configurable Several languages User interface is easy to configure

Table 3.2: Bug Tracking open source solutions

List of discontinued bug tracker projects:

- BugIn'Ticketing System: built in PHP and MySQL database, last release: March 2004;
- Gnatsweb: Coded in Perl, last release: July 2003;
- Anthill Bug Manager: built in Perl and using MySQL database, last release: December 2002;
- BugBye: using c# language, last release: June 2003.

3.3.3 Bug trackers functionalities

The main features of bug trackers are:

- **Change history**
System must store all issue state changes. Auditing may need this data and it is easy to know who closed the issue and how long it takes to solve it.
- **Configurable custom fields**
The system should have the ability to define custom fields by project. For some projects there are fields that should always be filled. For other projects these fields do not make sense, so configuration is needed.
- **Easy to use**
The system should be easy to use, because it is important that users do not give up issue submission. The web interface is a forward step because it avoids any problems in application installing, maintaining and configuring.

- **Email notifications**

The system should send emails when the issues change from state, priority or are assigned. This configuration must be flexible, per project and state. When an issue is assigned to a team element, he should be warned by email that an issue requires his attention to be solved.

- **Security**

The systems have integration with existent authentication systems such as LDAP or Active Directory. This feature enables existing user configuration reuse. There is no need to configure the users login and passwords across different systems, as the configuration is centralized in LDAP. Users must be aggregated into groups and must be possible to configure permissions by:

- Project
- Issue state change
- Fields visualization: *eg* end users cannot see technical discussions between helpdesk and programmer.
- Reports: systems produce reports about open issues, closed issues and resolution times. Exporting reports to Excel is usually supported.

- **Workflow**

Systems have predefined workflows and possibility to configure a new workflow. In addition, they provide a way to link some states with users and assign automatically the issues.

- **Multiple access detection**

Systems must be capable to identify and prevent multiple accesses to the same database register. System must warn user that the register was changed since last view.

- **Unicode support**

Unicode support is very important to ensure that other language characters are supported.

- **URI (Uniform Resource Identifier) by Issue**

Systems usually support URI (RFC2396). This is useful to enable users to bookmark direct access to the issue description/history without browsing through the web interface.

- **Search**

Free or by field values, search is a required feature for all tools aiming to address the issue tracking.

- **File attachment**

File attachment is essential because sometimes it is necessary to provide images or other files to show the occurring problem.

3.4 Commercial versus Open Source bug trackers

Commercial and open source bug tracking software are comparable in features. The main differences when comparing both approaches are:

- **Support**

Commercial solutions typically have support lines and bug fixes for urgent cases.

Open source software, does not guarantee support. This is provided through on-line forums where specialists share and solve issues inside the software.

To ensure proper support, companies usually contract individuals specialized in the software or contract other companies to get the operational support.

- **Access to code**

The access to code is an advantage of open source software. Commercial solutions usually do not share the code.

Not having access to code may be catastrophic when a software producer goes into a bankruptcy and the user needs updates or improvements.

Another big problem for government and organizations is to ensure that the bought software does not have any suspicious part.

- **Improvements**

Usually the open source community creates more improvement releases than private companies do.

Despite this constant delivery of new features, sometimes the features needed by some company are not made because they are not part of the roadmap or they are not generic to all community.

Of course, each company could use the base software and add the features needed.

Each company needs to evaluate which solution fits better their business. Both approaches have advantages and disadvantages and both have space

to coexist.

3.5 Using a bug tracker as base for development

The main factors to choose an open source bug tracker to base Amplidir development were:

- Application must be web based;
- Application must run on top of open source platform, such as:
 - Apache web server;
 - MySQL database;
 - Open source web scripting language, for example: PHP.

Taking these pre-requirements, we analyzed the following software packages:

- Mantis Bug Tracker ⁶
- phpBugTracker ⁷

After some study around installation and code analysis, we decided for “Mantis BugTracker”, with the following arguments:

- Better organization: software is split in different libraries to access different entities of the system, such as: bugs, users, groups, attachments...
- Library framework: the existent library enables a fast module creation, because the database access is already implemented through class and methods.

The main features that got us in deciding for the “Mantis BugTracker” were:

- Web based;
- Supports any platform that runs PHP (Windows, Linux, Mac, Solaris, AS400/i5, etc);
- Available in 68 localizations;
- Simple/advanced issue pages;

⁶<http://www.mantisbt.org>

⁷<http://phpbt.sourceforge.net/>

- Issues change log;
- Email notifications;
- Users can monitor specific issues;
- Attachments (can be saved on web server or in the database - can also backup to an FTP account);
- Issue change history;
- Issue relationships;
- Multi-DBMS Support:
 - MySQL;
 - MS SQL;
 - DB2;
 - PostgreSQL.

Chapter 4

Prototype Design

4.1 Prototype objectives

The Amplidir prototype used the Mantis bug tracker as a development base and addressed the following objectives:

- Authentication of users;
- Bug to issues and project to areas transformation;
- Issue submission and management;
- Questions submission and answers collecting;
- Voting system and multiple voting types;
- Issue “tagging”: users will be able to agree or disagree on issue notes, creating an indicator for the most popular notes.

4.2 Prototype features description

The features described above have a subset of features or requirements to implement. Next sections describe them.

4.2.1 Users authentication

The system should accept user’s authentication and allow organization by area. Mantis bug tracker already has this feature implemented. Anonymous access to system should not be allowed.

4.2.2 Bug to issues and project to area transformation

The main concept to reuse bug trackers software is to get the “bug/project” architecture and transform it into the “issue/area” concept.

The area concept replaces the project concept. All references to software projects (releases, operating systems, databases, versions) should be removed.

4.2.3 Issue submission and issue management

Users must be able to submit issues by areas and area owners should be able to manage an issues’s state, assign to users and request more information. Areas should hold whatever information is needed. System features for the issue submission:

- Creation of new areas and area permission management
- Submission of issues in area with permissions
- Posting comments on issues. Some user shall be able to comment any issue (on areas with permission), adding more information about it.
- Issues must have a state with possible values:
 - new
 - feedback
 - verified
 - assigned
 - accepted
 - closed (decided)
- Issues can be assigned to a manager
- System must be able to create a link (relation) between issues.
- Issues must have a history, with the following tracked actions:
 - Submission
 - Notes added
 - Notes deleted
 - Prioritization change
 - Assigning change

- System must permit configuration to send emails for area managers on new issue submission.
- System must be able to store file attachments related to issues.
- Free search in issues

4.2.4 Questions submission and answers collecting

System must give the ability for area members to create questions. Users must be able to answer the open questions. Features to be available:

- Create question: all members can create questions. Managers have access to advanced answer types.
- Questions must have the attributes:
 - Owner
 - Summary
 - Description
 - Type
 - Answer type
 - Date submitted
 - Date to close
 - Last update
- Questions have two states
 - Open: answers are accepted;
 - Closed: answers are not accepted.
- Questions have two types:
 - Authenticated: The identity of the question owner is shown;
 - Anonymous: The identity of the question owner is not shown to users.
- Question must have different answers types:
 - Forum mode: All answers are shown as they are submitted;
 - Advanced modes available to managers:
 - * Brainstorm mode: all answers are shown only when process finish;

- * Open Brainstorm mode: answers are shown, but without identification. Identification is showed at the end of process;
 - * Inquire mode: answers are never shown. Only question owner can see the answers.
-
- Question list view by area
Users must only view or access questions lists belonging to areas with permission.
 - Email notification
Users must be notified when a new question is submitted or a question is assigned.
 - Free search in questions and answers
Users must be able to search through the questions and answers they have permission to access.

4.2.5 Voting system

System must give managers the ability to create polls on the areas they manage. Users must be able to vote on options in open polls. Main features of the prototype:

- Poll creation;
System must able to create a poll in an area with a group of options to vote on.
- Polls must have the following attributes:
 - Owner
 - Summary
 - Description
 - Type
 - Voting type
 - Date submitted
 - Date to close
 - Last update
 - Options
 - Comments allowed
- Polls have two states;

- Open: voting is accepted;
- Closed: voting is not accepted.
- Polls have the following types:
 - Secret voting: the results are only shown in the end of process, without identities;
 - Anonymous open voting: results are shown as they are submitted, without identities;
 - Identified open voting: results are shown as they are submitted, with identification of the voters;
 - Anonymous on close voting: results are only shown in the end of process, with identification of the voters;
 - Inquire voting: results are never shown. Only poll owner can see the votes and identity.
- Polls may have the option for users to comment the poll, results, etc;
- The poll can be deleted by the poll owner;
- The poll cannot be edited after votes are being submitted;
- Poll list view by area;

Users must only access polls belonging to areas with permission.
- Email notification;

Users must be notified when a new poll is submitted.
- Free search in polls and comments.

Users must be able to search through the polls and comments they have permissions for.

4.2.6 Issue “tagging”

Users must be able to agree or disagree on the issue comments. Every comments on an issue should have a voting link; users could use this link to vote.

This feature will provide comments classification. Comments with more votes could show or indicate the best option. The identity of the user who vote in each comment is not public, so only the counters are available.

4.3 Design proposal for a Social Decision Support System - Amplidir

4.3.1 Assumptions and dependencies

As already mentioned the system will reuse the existent Mantis bug tracker source code (open source).

The system will run on the following platform:

- PHP 5
- Apache 2
- MySQL 5.0

The system can run wherever the Apache and PHP are supported, for example:

- Linux
- Windows
- Solaris

Users will use a web browser to access the system (Firefox or IExplore). The system will ask for identity and will store the sessions for the next connections.

4.3.2 Architectural strategies

Strategies to follow:

- **Programming language**

PHP to be used as the developing language. The reason for this decision is that the base software source code uses this language.

- **Database**

The database to use is the MySQL database. The reason for this decision is that the base system uses it and it is open source.

- **Web server**

The Web server to be used is the Apache 2, because it is the best web server to run the PHP runtime software.

- **ADODB Library**

Use of the ADODB library because the existent system uses it to serve as a wrapper interface to the database, providing some useful functions.

- **Mantis source code**

Reusing Mantis source code to implement issue, areas and permissions
The new features will be added on top of the Mantis system reusing the area/ permissions schema.
New requisites will imply database schema changes.

- **Expandable in mind**

This prototype design will be used to test the acceptance of SDSSs.
The implementation will take as an objective the possibility to expand in future developments.

- **User interface**

User interface will be web based and will be as simple as possible, using only standard html format and javascript.

- **Database errors**

User will receive the database errors with detailed description about the error, because this system is targeted to run inside organizations and not on open web, where such detailed descriptions could be used to develop exploits to the system. This behavior must be configurable.

- **System information**

The database will contain all the data and configurations for the system.

The only exception are the attachment files, which will be stored in separate file system. This decision has database performance in mind.

4.3.3 System architecture

Figure 4.1 shows the system components and the relation between the parts.

The areas will be the aggregating component. Users, issues, questions and polls will belong to areas. Issues will have notes sent by users. Questions will have answers submitted by users

Questions/answers and voting features was a new design. The issues part was done reusing the bug tracking existing architecture for bug tracking.

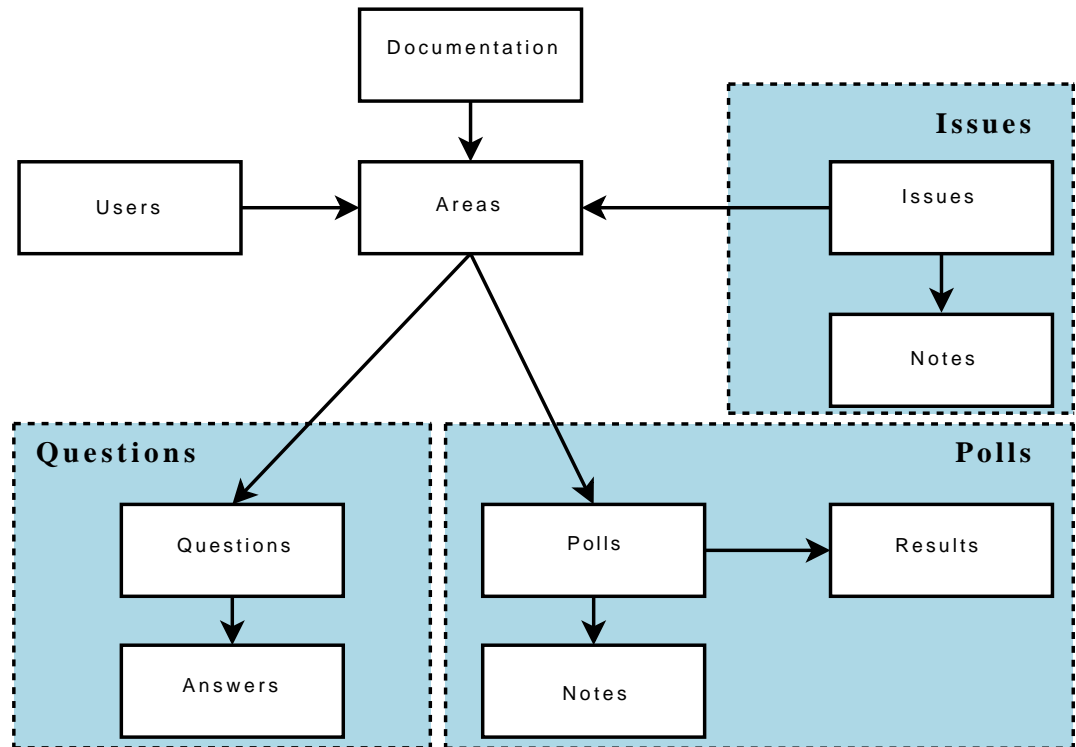


Figure 4.1: System overview

4.3.4 Questions and answers

System actions

Actions to implement:

- Create question
- Modify question status
- Delete question
- Question list for an area
- Question information view
- Add an answer to a question
- Search through questions and answers

User interface source code files description

The files in Table 4.1 were developed to create the actions described:

File name	Description
view_all_question_page.php	List all questions for the selected area
view_all_question_inc.php	Code responsible for the user interface creation for the question list view.
question_view.php	Show description and answers for the selected question
question_view_inc.php	Code responsible for the user interface creation for the question view.
question_create.php	Process the question creation submission by the users
question_change.php	Process the question status and other items change submission by the users
answer_add.php	Process the answer submission by the users
answer_add_inc.php	Code responsible for the user interface creation for the answer form.
answer_view_inc.php	Code responsible for the user interface creation for the answers list.

Table 4.1: User interface source code files description

Library source code files description

The files in table 4.2 were developed to support the user interface actions:

File name	Description
question_api.php	Collection of functions to deal with questions database tables and gathering information.
questionanswer_api.php	Collection of functions to deal with answers database tables and gathering information.

Table 4.2: Library source code files description

Classes, functions and database schema defining Questions

The class definitions, functions and database schemas are defined in Appendix A.1.

4.3.5 Voting system

System actions

The follow actions will be supported:

- Manager Actions
 - Create poll
 - Modify poll status
 - Delete poll
 - View poll results (non-public)
- User actions
 - Question poll list for an area
 - Poll information view
 - Add a vote to a question
 - Add comment to poll
 - View poll results
 - Search through polls and comments

Voting system user interface source code files description

The files in table 4.3 were developed to create the voting system described.

File name	Description
view_all_poll_page.php	List all poll for the selected area
view_all_poll.inc.php	Code responsible for the user interface creation for the poll list view.
poll_view.php	Show description and answers for the selected question
poll_view_inc.php	Code responsible for the user interface creation for the poll view.
vote_add.php	Process the vote submission by the users
vote_add.inc.php	Code responsible for the user interface creation for the vote form.
vote_view_inc.php	Code responsible for the user interface creation for the votes list.
poll_add_comment.php	Process the comment submission by the users on a poll

poll_create.php	Process the poll creation submission by the users
poll_change.php	Process the poll status and other items change submission by the users

Table 4.3: Voting system user interface source code files description

Voting System library source code files description

The files in table 4.4 were developed to support the user interface actions:

File name	Description
poll_api.php	Collection of functions to deal with poll database tables and gathering information.
poll_voting_api.php	Collection of functions to deal with votes database tables and gathering information.
poll_comments_api.php	Collection of functions to deal with poll comments database tables and gathering information.

Table 4.4: Voting system library source code files description

Classes and functions to be developed in poll_api.php

The classes, functions and database schema is described in appendix B.1.

4.3.6 Issue tagging

Functions and database schema to support the issue tagging feature can be found in appendix C.1.

4.3.7 Detailed system design: action diagrams

Diagrams with detailed action flow are in Appendix D.1.

Chapter 5

Prototype Implementation

5.1 Amplidir prototype implementation

The Amplidir prototype implementation followed some basic ideas:

- Extreme Programming principles¹
The main idea to follow from these principles was to start simple and add functionalities.
- Keep existing abstraction layers
Prototype should maintain Mantis bug tracker abstraction layers and reuse these layers wherever possible, without creating new layers for existing data or GUI structures.

Despite the existing powerful interface, it was needed to rework some GUI parts, because the GUI complexity for dealing with bug tracking is exaggerated for the Amplidir prototype.

Therefore the search GUI page and engine was re-made to be less complex, more intuitive and less dependent of the bug tracking concept. Listings were also re-done to create a cleaner interface. Figures 5.1 and 5.2 show a Mantis bug list and a clean issue list in Amplidir.


5.1.1 Development environment

The development environment used was the 'Wampserver'². This software creates an instance of:

- Apache 2.2.8

¹http://www.xprogramming.com/what_is_xp.htm

²<http://www.en.wampserver.com/>



Anonymous | [Login](#) | [Signup for a new account](#) 2008-07-02 17:42 CDT Project: [All Projects](#) [Switch](#) [Help](#)

[Main](#) | [My View](#) | [View Issues](#) | [Change Log](#) | [Roadmap](#) | [Docs](#) | [Wiki](#) | [Billing](#) Issue # [Jump](#)


Reporter:	Monitored By:	Assigned To:	Category:	Severity:	Resolution:	Profile:
any	any	any	any	any	any	any
Status:	Hide Status:	Product Build:	Product Version:	Fixed in Version:	Priority:	Target Version:
any	closed (And Above)	any	any	any	any	any
Show:	View Status:	Show Sticky Issues:	Changed(hrs):	Use Date Filters:	Relationships:	
50	any	Yes	6	No	any	
Platform:	OS:	OS Version:	Tags:			
any	any	any				
Sort by:	Last Update Descending					

Search: [Apply Filter](#) [Advanced Filters](#) [Create Permalink](#) [Reset Filter](#) [Use Filter](#) [Manage Filters](#)

Viewing Issues (1 - 50 / 777) [Print Reports](#) [CSV Export](#) [First Prev 1 2 3 4 5 6 7 8 9 10 11 ... Next Last]

	P	ID	US\$	#	@	Category	Severity	Status	Updated	Summary
		0003746				[Demo] GUI	minor	assigned (gpiras)	2008-07-02	[MANGA]: nuovo backdrop
		0003723				[Demo] Other	crash	assigned (klakla)	2008-07-02	It's impossible to work with this system ;-)
		0003052		1		[Demo] Other	minor	assigned (test)	2008-07-02	e
		0003745				[Demo] GUI	minor	assigned (hans1234)	2008-07-02	test
		0003744				[Demo] GUI	minor	assigned (msatdh)	2008-07-02	C-Store Queue display flickers
		0003743				[Demo]	minor	assigned (msatdh)	2008-07-01	Test

Figure 5.1: Mantis list



DEEIC - Amplidir Development Group

Utilizador: Nelson Faria


[Principal](#) | [Questões](#) | [Perguntas](#) | [Inquéritos](#) | [Gestão](#) | [Sair](#)

Amplidir >> Todas as Areas >> Questões

Questões (1 - 10 / 12) [Imprimir Relatório de Questões](#) [Exportação em formato CSV](#) [Primeiro Prev 1 2 Prox Ultimo]

Título	Area	Prioridade Inicial	Utilizador	Estado	Última Atualização	#Notas
questao de teste	New Instance	Normal	nfaria	Nova	2008-07-27 22:38:30	2
eryr	amplidir_development	Normal	nfaria	Nova	2008-07-02 23:52:26	
test	amplidir_development	Normal	nfaria	Nova	2008-07-02 23:35:48	
Notas de reuniao 2008.04.22	amplidir_development	Normal	nfaria	Nova	2008-06-02 22:35:00	2
Notas de reuniao - Alterações necessarias	amplidir_development	Normal	nfaria	Destinada	2008-04-15 23:26:31	4
FreeBoard beta version	amplidir_development	Normal	nfaria	Destinada	2008-04-05 10:36:18	3
marcação de telecon	amplidir_development	Normal	nfaria	Resolvida	2008-03-23 22:25:21	2
Reuniao 2008.03.15	amplidir_development	Normal	nfaria	Nova	2008-03-23 11:49:27	1
Termos técnicos	amplidir_development	Alta	pgarrido	Nova	2008-03-06 21:44:31	1
Problema no acesso ao servidor	amplidir_development	Normal	nfaria	Encerrada	2008-02-25 08:42:19	3

Utilizador: Nelson Faria



Universidade do Minho

Figure 5.2: Amplidir issue list

- MySQL 5.0.51b
- PHP 5.2.6

Wampserver is a Windows web development environment. It allows to create web applications with Apache, PHP and the MySQL database. It also comes with PHPMyAdmin and SQLiteManager for easily managing the database. These software packages are the pre-requisites to run the Mantis bug tracker and the Amplidir prototype. The operating system used was Windows XP.

5.1.2 Test environment

The test environment used was a GNU/Linux server with:

- Apache2 Server
- MySQL
- phpMySQL

5.2 PHP Language Overview

5.2.1 PHP Introduction

PHP (recursive acronym for “PHP: Hypertext Preprocessor”) is an open source scripting language used for web development. The run-time engine run on top of web servers (Apache, Microsoft IIS) and has wide support for common technologies as:

- Multiple database support:
 - Adabas D
 - dBase
 - Empress
 - FilePro (read-only)
 - Hyperwave
 - IBM DB2
 - Informix
 - Ingres
 - InterBase
 - FrontBase

- mSQL
 - Direct MS-SQL
 - MySQL
 - ODBC
 - Oracle (OCI7 and OCI8)
 - Ovrimos
 - PostgreSQL
 - SQLite
 - Solid
 - Sybase
 - Velocis
 - Unix dbm
- Output XHTML, XML, text, images, PDF files and even Flash movies
- Support for talking to other services using protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM and CORBA
- XML documents processing using the SAX and DOM standards
- Compression utilities (gzip, bz2, zip)
- Authentication services: KADM5 : Kerberos V and Radius
- Calendar conversions
- Credit card processing: MCVE - MCVE (Monetra) Payment and SP-PLUS - SPPLUS Payment System
- Cryptography extensions
 - Cracklib
 - Hash HASH Message Digest Framework
 - Mcrypt
 - Mhash
 - OpenSSL
- Web services
- SCA
- SOAP

- XML-RPC

The code can be mixed with HTML to create output format. Next code listing is a "Hello World" script in PHP:

```
<html>
<head>
  <title>PHP Test</title>
</head>
<body>
  <?php echo '<p>Hello World</p>'; ?>
</body>
</html>
```

The code is separated from the normal HTML by the strings: '<?php' and '?>'.

The language is simple and powerful and supports advanced programming topics, such as function parameters by reference, multidimensional arrays, or even a class concept. In Appendix E.1, a PHP language introduction is included.

5.3 Implementation approach

The implementation followed some of the Extreme Programming principles, such as:

- **Simplicity**

Start with a simple solution first and add more advanced features later.

This approach has some advantages which help to get fast software development:

- Early availability of software to test and collect feedback (not a final product). This is useful to fine-tune the software design and even to redo some parts;
- Time is not spent in constructing software for a future where its specifications could change dramatically;
- Better alignment between software construction and final user, because it is tested during the development;
- Feedback.

The system was tested and reviewed several times during implementation. In each review, an issue list was produced and the issues corrected or changed into next review.

- **Courage**

The principle of “simplicity” sometimes has a big drawback. Produced code fully working for the first approach becomes obsolete for the next release. Some courage is needed to throw away code where a significant amount of time was invested. During this project, some parts were redone, for example: listings and layouts.

Using these principles, it was decided to follow the plan:

- Put Mantis (software base) working;
- Put this software working in an available server;
- Change the Mantis software, converting bugs in issues and projects in areas;
- Implement new functionalities;
- Perform testing;
- Review, providing feedback;
- Rewrite or modify code;
- Test, review, modify.

The software construction followed the principle of bottom up integration. The lower software levels are constructed and integrated first (*i.e.* data access functions and logic functions). The last layer to be integrated was the user interface.

The main advantages of this approach are:

- Easier to create test cases and observe output;
- No stubs are required;
- Errors in critical modules are found early;
- It supports reuse of low-level units;
- Interface faults can be easily found. When developers substitute a test driver by a higher level component, they have a clear model of how the lower level component works and of the assumptions embedded in its interface.

5.4 Changes in the software design

Some features and code available in the prototype were not part of the specification. These features resulted from reviews, tests and implementation needs. It was assumed that the specification was a start statement and changes were to be made along the implementation process as needed.

The features added were:

- Subfunctions included into the existing library to simplify interfaces;
- The search functionality was created for issues and questions.

The existing interface and code was too complex and with several filters. The prototype needed a clean interface. The search functionality was reduced to one text field search;

- Changes since last visit

One functionality desired for the prototype was overviewing what was created or updated since last login. The main page was used to show the user all the issues, question and polls updated since last visit. To implement this feature, Mantis library Tokens API was used. Basically this API let us save in the database per user settings. The last visit is saved and used in the login to query the system for the updated issues or questions.

5.5 Help material to users

Most important characteristic of software is its usability. A facility for introducing multi-lingual mouse sensitive help text was developed. An introductory text to the Amplidir concept is available as follow:

AMPLIDIR

AMPLIDIR is a social decision support system for organizations, based upon collective intelligence results. The aim of AMPLIDIR is to enable organizations to explore the knowledge that every people in the organization have about the organization itself, in order to better the quality of decisions.

Under AMPLIDIR, decision processes, being the responsibility of managers, are conceived to be distributed along all the people in the organization, in a way that is not necessarily explicit. Basically, AMPLIDIR is a management system for messages relevant to decision processes that people exchange. The system is designed having in mind to allow making explicit

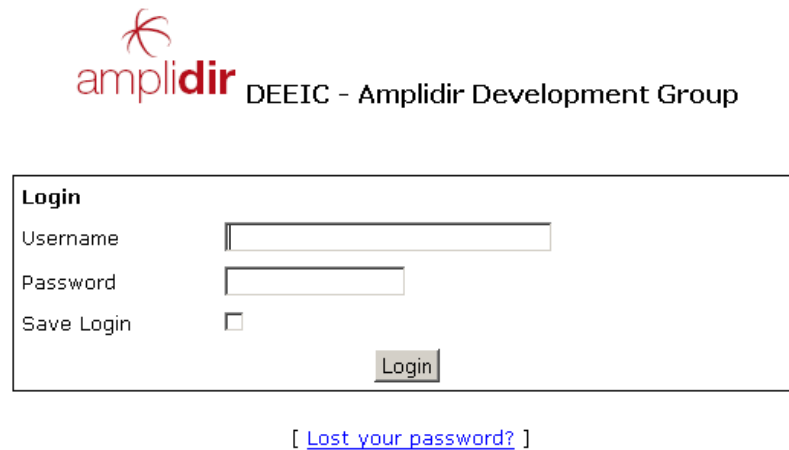
the knowledge and perceptions people have, creating an amplified social support for decision.

AMPLIDIR supports assigning people to (possibly several) areas in a very flexible way in order to mirror the organizations static and dynamic structure. Within AMPLIDIR, it is possible for every people in the organization to pose issues and questions relevant to decisions. Fast feedback mechanisms allow managers to assess quickly the social relevance or support for the issues posed. It is also possible for managers to get information on demand from other people to base decisions, with some sophisticated query and voting mechanisms.

5.6 Prototype Interface

5.6.1 Login page

The login page requests a login and password for user authentication. A link is available to request an automated password recovery. See Figure 5.3.



amplidir DEEIC - Amplidir Development Group

Login

Username

Password

Save Login ☐

Login

[[Lost your password?](#)]

Figure 5.3: Login page

5.6.2 Entry page

The main page presents the user with some useful information (Figure 5.4):

- Last news edited by area manager;

- A fast menu access to system components;
- The issues / questions and polls created or updated since last visit;
- Area selection on left of the page;
- Information below the menu with the current location in the system.

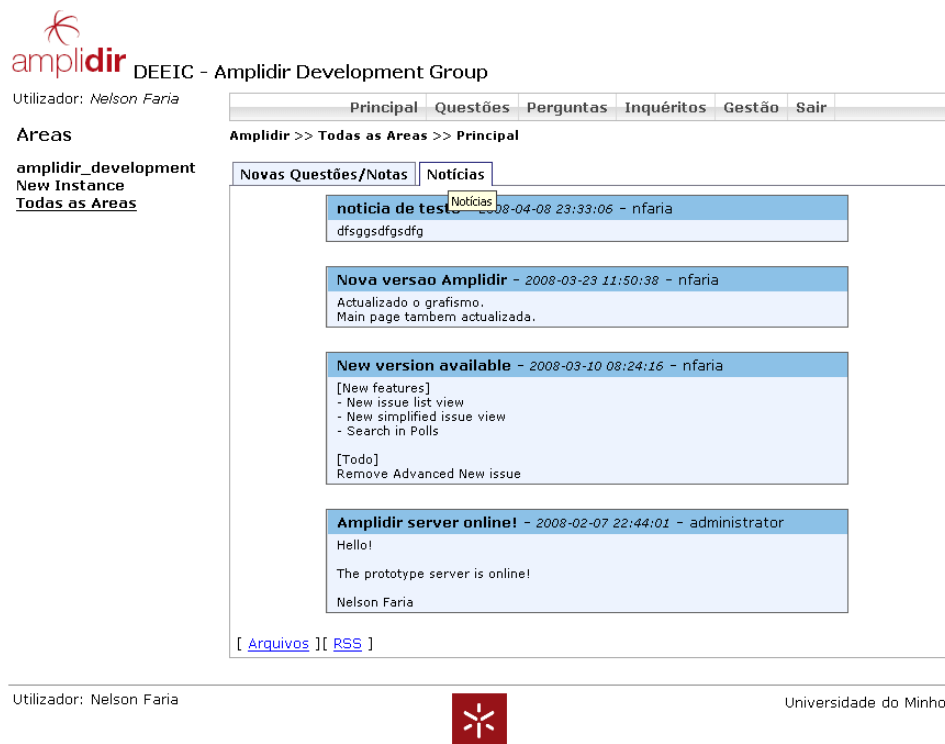


Figure 5.4: Entry page

5.6.3 Main Menu

The main menu is a simplified drop down menu for access to the prototype system components (Figure 5.5):

- Issues;
- Questions;
- Polls;
- Management.

For issues, questions and polls, the organization is always the same:

- All the topics(issues/questions/polls);
- Create a new topic;
- The topics initiated by me;
- Search the topics.

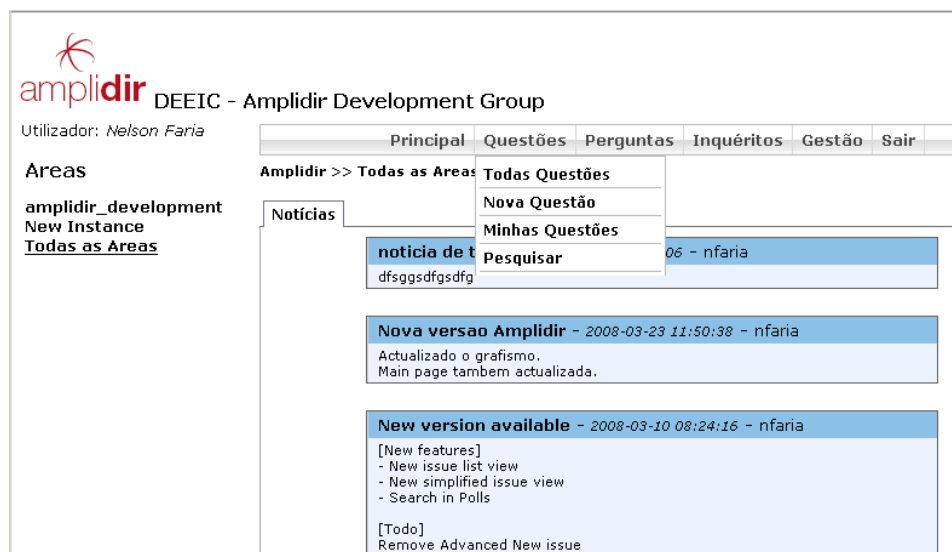


Figure 5.5: Main menu

5.6.4 Issue List

The issue list shows issues ordered by last update time stamp, but is possible to order the issue list by different fields criteria. Please refer to Figure 5.2.

5.6.5 Issue comments tagging

It is possible to agree or disagree not only with the issue, but also with the submitted comments. This gives community members the feedback of the individual. See Figure 5.6 with the interface detail for comment tagging.

Notas da Questão	
(0000026) nfaria 2008-04-05 11:00:54 ↑(1) ↓(0)	uma nota de teste
(0000033) nfaria 2008-07-27 22:38:30 ↑(0) ↓(0)	Apenas uma resposta de teste ↑ Concordo com esta nota ↓ Discordo desta nota

Figure 5.6: Issue tagging

5.6.6 Issue fast feedback

Usability is a key success factor. Fast shortcuts to be used on repetitive actions are a way to achieve it. The fast answer feedback is provided to reply in a fast way to simple issues. See Figure 5.7.

Nota Rápida:	Concordo	Não Concordo	Subscrevo	Não Subscrevo	Confirma	Não Confirma
--------------	----------	--------------	-----------	---------------	----------	--------------

Figure 5.7: Fast feedback

5.6.7 Issue searching

The searching interface is clear and simple. A simple text field is presented to the user. After one word or more are inserted, the issues having matching text are presented. Figure 5.8 shows the searching interface and figure 5.9 is the image of a performed search.

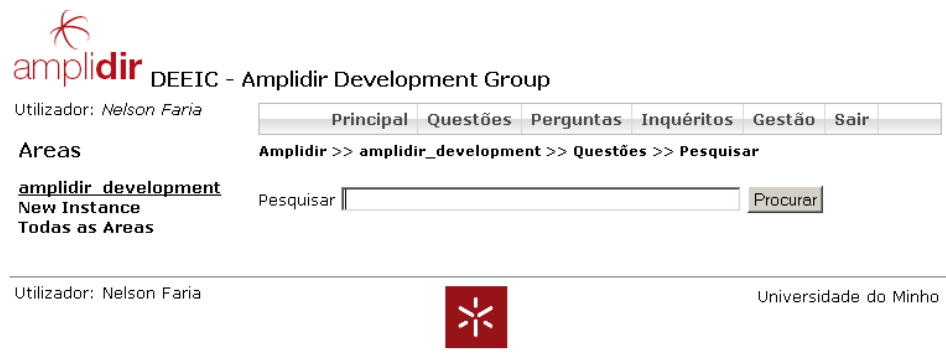


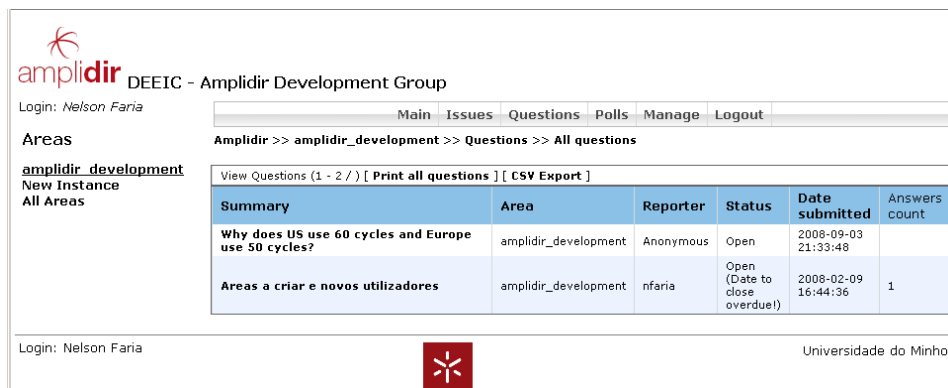
Figure 5.8: Issue search



Figure 5.9: Issue search results

5.6.8 Question list

An interface similar to the issue list was created for question listing. Please see figure 5.10



amplidir DEEIC - Amplidir Development Group

Login: Nelson Faria

Main Issues Questions Polls Manage Logout

Areas

amplidir_development
New Instance
All Areas

Amplidir >> amplidir_development >> Questions >> All questions

View Questions (1 - 2 /) [Print all questions] [CSV Export]

Summary	Area	Reporter	Status	Date submitted	Answers count
Why does US use 60 cycles and Europe use 50 cycles?	amplidir_development	Anonymous	Open	2008-09-03 21:33:48	
Areas a criar e novos utilizadores	amplidir_development	nfaria	Open (Date to close overdue!)	2008-02-09 16:44:36	1

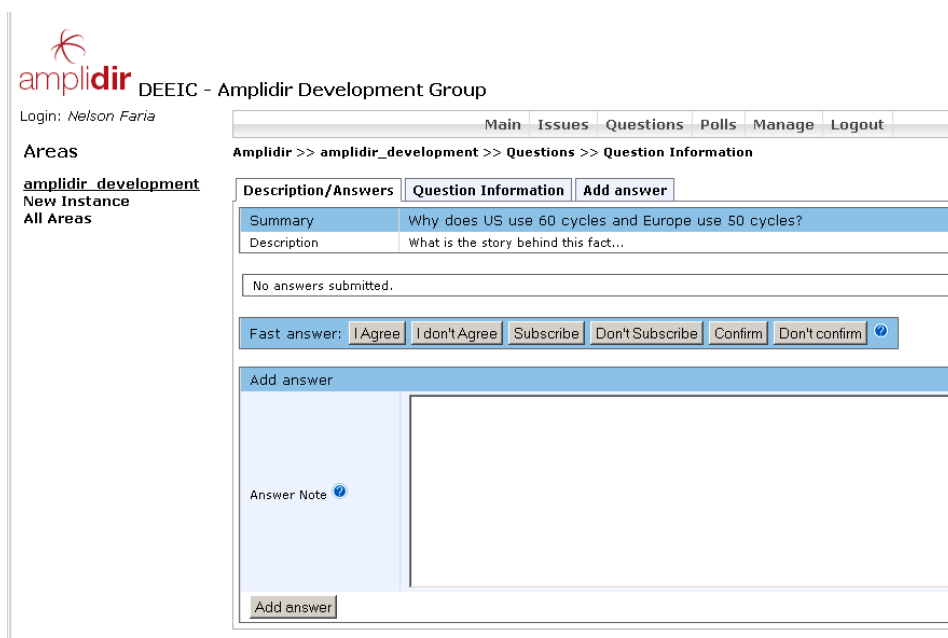
Login: Nelson Faria

Universidade do Minho

Figure 5.10: Question list

5.6.9 Question detail

The question detail interface gives access to the question detail and if the question is open, members can reply immediately, as shown in figure 5.11



amplidir DEEIC - Amplidir Development Group

Login: Nelson Faria

Main Issues Questions Polls Manage Logout

Areas

amplidir_development
New Instance
All Areas

Amplidir >> amplidir_development >> Questions >> Question Information

Description/Answers Question Information Add answer

Summary	Why does US use 60 cycles and Europe use 50 cycles?
Description	What is the story behind this fact...

No answers submitted.

Fast answer: I Agree I don't Agree Subscribe Don't Subscribe Confirm Don't confirm

Add answer

Answer Note

Add answer

Figure 5.11: Question detail

5.6.10 Question submission

Figure 5.12 shows the interface to submit a new question. Some configurations can be made at this point, such as question type, answer type, date to close

amplidir DEEIC - Amplidir Development Group

Login: Nelson Faria

Main Issues Questions Polls Manage Logout

Areas

Amplidir >> amplidir_development >> Questions >> Submit question

amplidir_development
New Instance
All Areas

Enter question details

*Summary

*Description

Question Type: Anonymous

Answer Type: BrainStorm mode: All answers are shown only when process finish

Date to close: 2008-10-03 21:46:07

* required Submit question

Login: Nelson Faria Universidade do Mir

Figure 5.12: Question submission

5.6.11 Poll list

Figure 5.13 shows the interface for the poll list is similar to issues and questions, maintaining a consistent look through the interface.

amplidir DEEIC - Amplidir Development Group

Login: Nelson Faria

Main Issues Questions Polls Manage Logout

Areas

Amplidir >> amplidir_development >> Polls >> All polls

amplidir_development
New Instance
All Areas

View polls (1 - 2 /) [Print all polls] [CSV Export]

Summary	Area	Reporter	Status	Date submitted	Votes count
Determine the color of the new product	amplidir_development	Anonymous	Open	2008-09-03 22:37:19	
1 Inquerito	amplidir_development	nfaria	Open (Date to close overdue!)	2008-03-15 12:13:50	

Login: Nelson Faria Universidade do Minho

Figure 5.13: Poll list

5.6.12 Poll detail, voting and results detail

Figures 5.14, 5.15 and 5.16 show the interface for poll details, to participate in a poll and viewing a poll results.



Figure 5.14: Poll detail

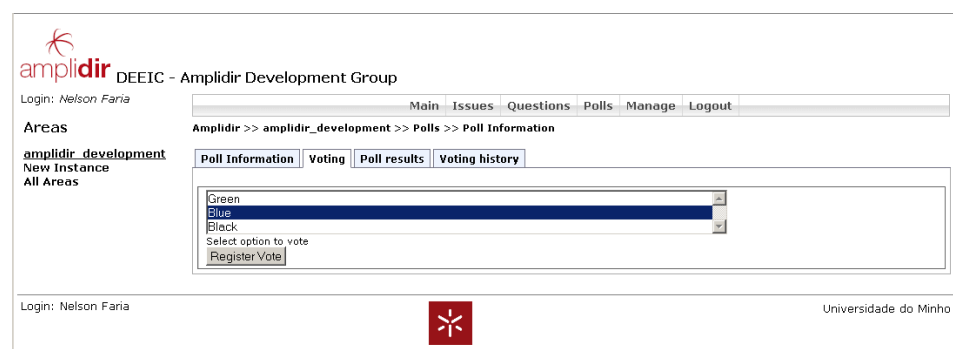
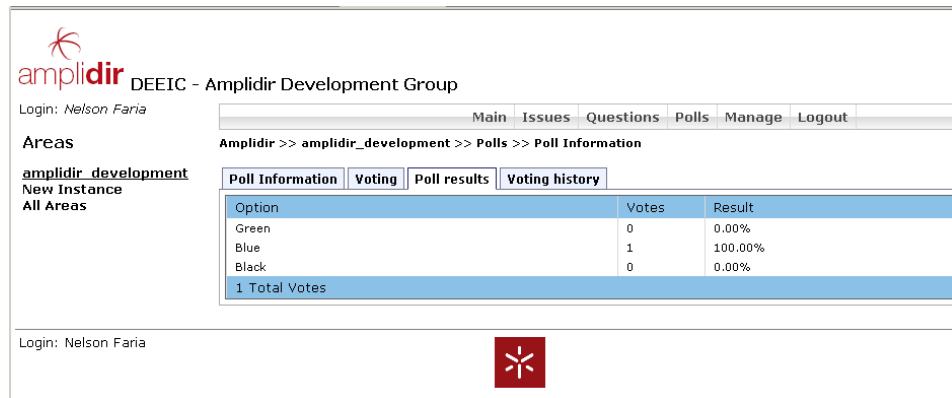


Figure 5.15: Poll voting



The screenshot shows the Amplidir web application interface. At the top, there is a logo for "amplidir" and the text "DEEIC - Amplidir Development Group". Below this, a navigation bar includes links for "Main", "Issues", "Questions", "Polls", "Manage", and "Logout". The user is logged in as "Nelson Faria".

The main content area displays the path "Amplidir >> amplidir_development >> Polls >> Poll Information". There are four tabs: "Poll Information", "Voting", "Poll results", and "Voting history". The "Poll Information" tab is active, showing a table with the following data:

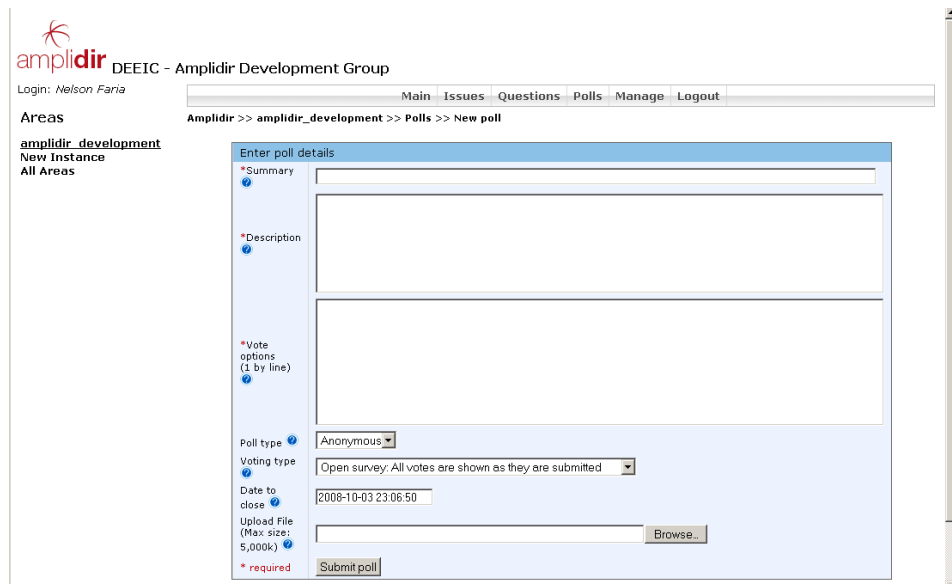
Option	Votes	Result
Green	0	0.00%
Blue	1	100.00%
Black	0	0.00%
1 Total Votes		

At the bottom of the page, there is a red square logo with a white star-like symbol.

Figure 5.16: Poll results detail

5.6.13 Poll submission

Figure 5.17 shows the interface for new poll submission.



The screenshot shows the "Enter poll details" form in the Amplidir web application. The form is titled "Enter poll details" and includes the following fields and options:

- *Summary**: A text input field.
- *Description**: A text input field.
- *Vote options (1 by line)**: A text input field for entering multiple options.
- Poll type**: A dropdown menu with "Anonymous" selected.
- Voting type**: A dropdown menu with "Open survey: All votes are shown as they are submitted" selected.
- Date to close**: A date input field showing "2008-10-03 23:06:50".
- Upload File (Max. size: 5,000k)**: A text input field with a "Browse..." button.
- *required**: A label indicating that the "Submit poll" button is required.

The "Submit poll" button is located at the bottom of the form.

Figure 5.17: Poll submission

5.7 Tests performed

On top of the development test cases, the system was used as a work group platform for the dissertation work. The reviews performed ensured testing and proper correction of bugs found.

During development, the database was filled with issues and notes by a test script to ensure the correct system response on the long run.

An open field test, with multiple instances of Amplidir and multiple users to collect user feedback and prepare next versions, is planned.

Chapter 6

Conclusion

6.1 Results

All the specified features for the Amplidir prototype were implemented. Some non-specified features were also implemented. The specified and implemented features were:

- Issues
 - Bug into issue and project into area conversion;
 - Redesign of all interface, making it simple;
 - Cleared all data logic not needed for the Amplidir (bug tracking concepts, i.e. version, release...);
 - Issue tagging interface and logic.
- Questions and answers
 - Database logic
 - User interface
- Polls
 - Database logic
 - User interface

The features not specified and implemented were:

- Interface for visualization of all issues updated or created since last login;
- Search routine for issues to simplify the existing search interface.

6.2 Advantages and disadvantages using the open source Mantis bug tracker

Usage of existing software had some important advantages:

- Already existing logic to interact with data;
- Initial effort to select and utilize external components was not necessary (i.e. database, authentication,...);
- Fast setup to start development and testing.

Despite the advantages, some drawbacks of the approach emerged as:

- Limitation in database growth, due to existing architecture;
- Difficulty to find tricky issues, due to the amount of code;
- Limitation in the platform to use;
- Limitation on the re-usage of components;
- Some new features could lead to a total rework of the modules;
- Side effects on reusing code that was used for more than one objective.

Despite the usage of Mantis did not show big disadvantages, some drawbacks affected the final prototype:

- Extra work to create a clear interface without the "bug tracking" concepts;
- Rework of search capability.

Balancing the pros and cons of the choice, the result was positive and enabled fast prototyping. There are some modules already available in the platform, which created a better finish prototype:

- Users authentication;
- Files upload;
- File storing in database;
- Email notification API;
- Administration tool;
- Multiple database support;
- Localization already implemented;

Summarizing, Mantis is a good project to be used as base for other PHP projects, due to common development issues are already resolved.

6.3 Future work

6.3.1 Field test of Amplidir

The system has now been developed to the point that it can be used in a real environment.

A field test must be conducted to collect users' feedback on implemented features and improvements needed.

6.3.2 New features to add

The prototype can grow in capabilities, heading to most important advanced features:

- Complex voting
Capture the confidence network and use it in complex voting schemes is a challenge.
It is necessary to create algorithms to verify that the voting represents the entire population and the confidence levels.
- Decision flow status
Implementation of a flow with the status of a decision process will enable all members to visualize what is under evaluation for decision and how the decision process advances.

Appendix A

Classes, functions and database schema to question API

A.1 Class QuestionData

Class QuestionData will be defined as follow:

```
class QuestionData {  
    var $id = 0;  
    var $project_id = 0;  
    var $reporter_id = 0;  
    var $text_id = 0;  
    var $status = 0;  
    var $summary = "";  
    var $question_type = 0;  
    var $answer_type = 0;  
    var $date_submitted = "";  
    var $last_updated = "";  
    var $date_close = "";  
}
```

A.2 Functions

- function question_get_question_data(\$p-question_id)
Function to return a class “QuestionData” filled with information about a question id

- function question_get_status_text(\$status)
Function to get the status text for a status integer
- function question_get_questiontype_text(\$type)
Function to get the question type text for a type integer
- function question_get_answertype_text(\$type)
Function to get the answer type text for a type integer
- function get_question_description(\$questionid)
Function to get the description text for a question (from table ad_question_text_table)
- function question_update_date(\$p_question_id)
Function to update last_update field for a question
- function question_update_status(\$p_question_id , \$status)
Function to update status field for a question
- function question_get_questions_list(&\$p_page_number, &\$p_per_page,
&\$p_page_count, &\$p_question_count, \$p_project_id = null)
Function to return a question list for an area using pagination
- function question_create(\$project_id, \$reporter_id, \$text, \$status, \$summary, \$ type, \$answer_type, \$date_close)
Function to create a new question, returns the question id.

A.3 Classes and functions to be developed in questionanswer_api.php

Class QuestionAnswerData will be defined as follow:

```
class QuestionAnswerData {
    var $id;
    var $question_id;
    var $reporter_id;
    var $answer;
    var $date_submitted;
    var $last_modified;
}
```

A.4 Functions

- function answer_exists(\$p_answer_id)

- function answer_ensure_exists(\$p_answer_id)
- function answer_is_user_reporter(\$p_answer_id, \$p_user_id)
- function answer_add (\$p_question_id, \$p_answer_text, \$p_private = false, \$p_user_id = null)
- function answer_delete(\$p_answer_id)
- function answer_delete_all(\$p_question_id)
- function answer_get_text(\$p_answer_id)
- function answer_get_field(\$p_answer_id, \$p_field_name)
- function answer_get_latest_id(\$p_answer_id)
- function answer_get_all_answers(\$p_question_id, \$p_user_answer_order, \$p_user_answer_limit)
- function answer_date_update(\$p_answer_id)
- function answer_set_text(\$p_answer_id, \$p_answer_text)
- function answer_format_id(\$p_answer_id)

A.5 Questions database schema

Next diagram shows the relations between questions and answers in database.

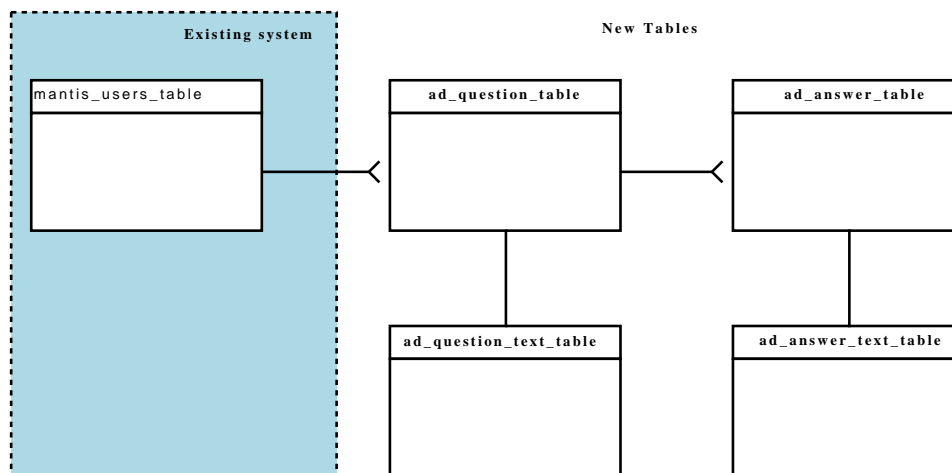


Figure A.1: Questions Database schema

SQL tables schema:

```
—
—Table structure for table 'ad_answer_table'
—

CREATE TABLE 'ad_answer_table' (
  'id' int(10) unsigned NOT NULL auto_increment,
  'question_id' int(10) unsigned NOT NULL default '0',
  'reporter_id' int(10) unsigned NOT NULL default '0',
  'answer_text_id' int(10) unsigned NOT NULL default '0',
  'date_submitted' datetime NOT NULL default '1970-01-01 00:00:01',
  'last_modified' datetime NOT NULL default '1970-01-01 00:00:01',
  PRIMARY KEY ('id'),
  KEY 'idx_question' ('question_id'),
  KEY 'idx_last_mod' ('last_modified')
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=18 ;

—
— Table structure for table '
—

CREATE TABLE 'ad_answer_text_table' (
  'id' int(10) unsigned NOT NULL auto_increment,
  'note' text NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=18 ;

—
— Table structure for table 'ad_question_table'
—

CREATE TABLE 'ad_question_table' (
  'id' int(10) unsigned NOT NULL auto_increment,
  'project_id' int(10) unsigned NOT NULL default '0',
  'reporter_id' int(10) unsigned NOT NULL,
  'question_text_id' int(10) unsigned NOT NULL default '0',
  'status' smallint(6) NOT NULL,
  'summary' varchar(128) NOT NULL,
  'question_type' smallint(6) NOT NULL default '0',
```

```
'answer_type' smallint(6) NOT NULL default '0',
'date_submitted' datetime NOT NULL default '1970-01-01 00:00:01',
'date_close' datetime NOT NULL default '1970-01-01 00:00:01',
'last_updated' datetime NOT NULL default '1970-01-01 00:00:01',
PRIMARY KEY ('id'),
KEY 'idx_question_status' ('status'),
KEY 'idx_project_id' ('project_id')
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=3 ;
```

```
—
— Table structure for table 'ad_question_text_table'
—
```

```
CREATE TABLE 'ad_question_text_table' (
  'id' int(10) unsigned NOT NULL auto_increment,
  'note' text NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=10 ;
```


Appendix B

Class, functions and database schema for polls API

B.1 Class PollData

Class PollData will be defined as follow:

```
class PollData{
    var $id = 0;
    var $project_id = 0;
    var $reporter_id = 0;
    var $text_id = 0;
    var $status = 0;
    var $summary = "";
    var $poll_type = 0;
    var $allow_comments=0;
    var $date_submitted = "";
    var $last_updated = "";
    var $date_close = "";
}
```

B.2 Functions

- function poll_get_poll_data(\$p_poll_id)
Function to return a class "PollData" filled with information about a question id
- function poll_get_status_text(\$status)
Function to get the status text for a status integer

- function poll_get_polltype_text(\$type)
Function to get the poll type text for a type integer
- function get_poll_description(\$poll_id)
Function to get the description text for a poll (from table ad_poll_text_table)
- function poll_update_date(\$poll_id)
Function to update last_update field for a question
- function poll_update_status(\$poll_id , \$status)
Function to update poll status
- function poll_get_poll_list(&\$p_page_number, &\$p_per_page, &\$p_page_count, &\$p_poll_count, \$p_project_id = null)
Function to return a poll list for an area using pagination
- function poll_create(\$project_id, \$reporter_id, \$text, \$status, \$summary, \$allow_comments, \$date_close)
Function to create a new poll, returns the poll id.
- function poll_get_options(\$poll_id, \$vote_active)
Function to create a view for option list, the form will be active if vote_active parameter is passed.

B.3 Functions to be developed in poll_voting_api.php

- function poll_voting_add(\$p_poll_id,\$reporter_id,\$option)
Add the vote from reporter_id on option poll.
- function poll_voting_get_total_votes(\$p_poll_id)
Function to return an array with total votes by option
- function poll_get_statistics(\$p_poll_id)
returns a statistics list (and graph) for a poll

B.4 Voting system database schema

Next picture s possible to see the Questions & answers relations database table diagrams.

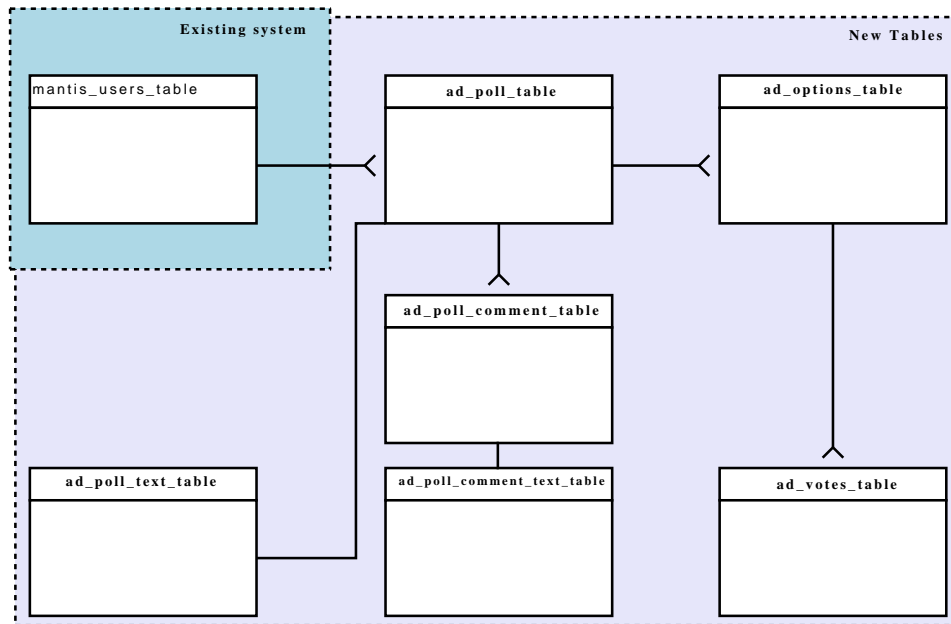


Figure B.1: Voting database schema

— Table structure for table 'ad_poll_table'

—

```
CREATE TABLE 'ad_poll_table' (
'id' int(10) unsigned NOT NULL auto_increment,
'project_id' int(10) unsigned NOT NULL default '0',
'reporter_id' int(10) unsigned NOT NULL,
'poll_text_id' int(10) unsigned NOT NULL default '0',
'status' smallint(6) NOT NULL,
'summary' varchar(128) NOT NULL,
'poll_type' smallint(6) NOT NULL default '0',
'allow_comments' smallint(6) NOT NULL,
'date_submitted' datetime NOT NULL default '1970-01-01 00:00:01',
'date_close' datetime NOT NULL default '1970-01-01 00:00:01',
'last_updated' datetime NOT NULL default '1970-01-01 00:00:01',
PRIMARY KEY ('id'),
KEY 'idx_poll_status' ('status'),
KEY 'idx_project_id' ('project_id')
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=3
;
```

```
—
— Table structure for table 'ad_poll_text_table'
—

CREATE TABLE 'ad_poll_text_table' (
'id' int(10) unsigned NOT NULL auto_increment,
'note' text NOT NULL,
PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=10
;

—
— Table structure for table 'ad_poll_comment_table'
—

CREATE TABLE 'ad_poll_comment_table' (
'id' int(10) unsigned NOT NULL auto_increment,
'poll_id' int(10) unsigned NOT NULL default '0',
'reporter_id' int(10) unsigned NOT NULL default '0',
'comment_text_id' int(10) unsigned NOT NULL default '0',
'date_submitted' datetime NOT NULL default '1970-01-01 00:00:01',
PRIMARY KEY ('id'),
KEY 'idx_poll_id' (poll_id),
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=18
;

—
— Table structure for table '
—

CREATE TABLE 'ad_poll_comment_text_table' (
'id' int(10) unsigned NOT NULL auto_increment,
'note' text NOT NULL,
PRIMARY KEY ('id'),
KEY 'idx_poll_id' (poll_id),
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=18
;

—
```

– Table structure for table ‘ad_poll_options_table’

–

```
CREATE TABLE ‘ ad_options_table‘ (  
‘id‘ int(10) unsigned NOT NULL auto_increment,  
‘poll_id‘ int(10) unsigned NOT NULL default ‘0’,  
‘option_text ‘ text NOT NULL,  
PRIMARY KEY (‘id’),  
KEY ‘idx_poll_id‘ (poll_id),  
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=18  
;
```

–

– Table structure for table ‘ad_votes_table’

–

```
CREATE TABLE ‘ ad_poll_option_table‘ (  
‘poll_id‘ int(10) unsigned NOT NULL default ‘0’,  
‘option_id‘ int(10) unsigned NOT NULL default ‘0’,  
‘reporter_id‘ int(10) unsigned NOT NULL default ‘0’,  
KEY ‘idx_poll_id‘ (poll_id),  
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=18  
;
```


Appendix C

Issue tagging functions and database schema

C.1 System actions

The following APIs will be developed to support the "Issue Tagging":

- function `note_vote($note_id, $type, $userid)`
Function to register a vote on a certain note.
- function `get_note_votes($note_id, $type)`
Function to get the votes on a note id for a type: agree or disagree
- function `already_note_voted($note_id, $userid)`
Function to call to enable the voting links for an issue to the current user.

C.2 Issue tagging database schema

The table to be created has a relation between a note and a user.

The user cannot vote twice for the same note.

The database schema is defined as follows:

```
CREATE TABLE 'mantis_ad_note_votes_table' (  
  'note_id' int(10) unsigned NOT NULL default '0',  
  'vote_type' smallint(6) unsigned NOT NULL default '0',  
  'reporter_id' int(10) unsigned NOT NULL default '0',  
  'vote_date' datetime NOT NULL,  
  PRIMARY KEY ('note_id','reporter_id')  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```


Appendix D

Detailed system design: action diagrams

D.1 Detailed system design: Questions action diagrams

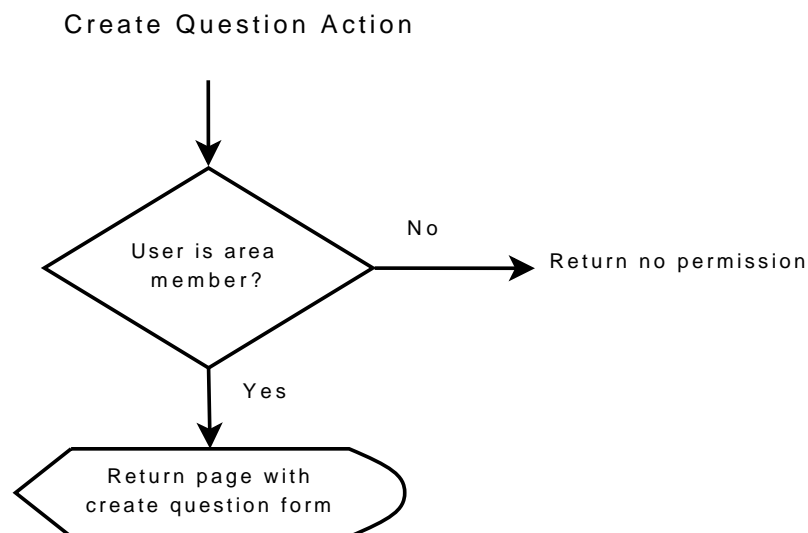


Figure D.1: Create Question

Create Question Action (submission)

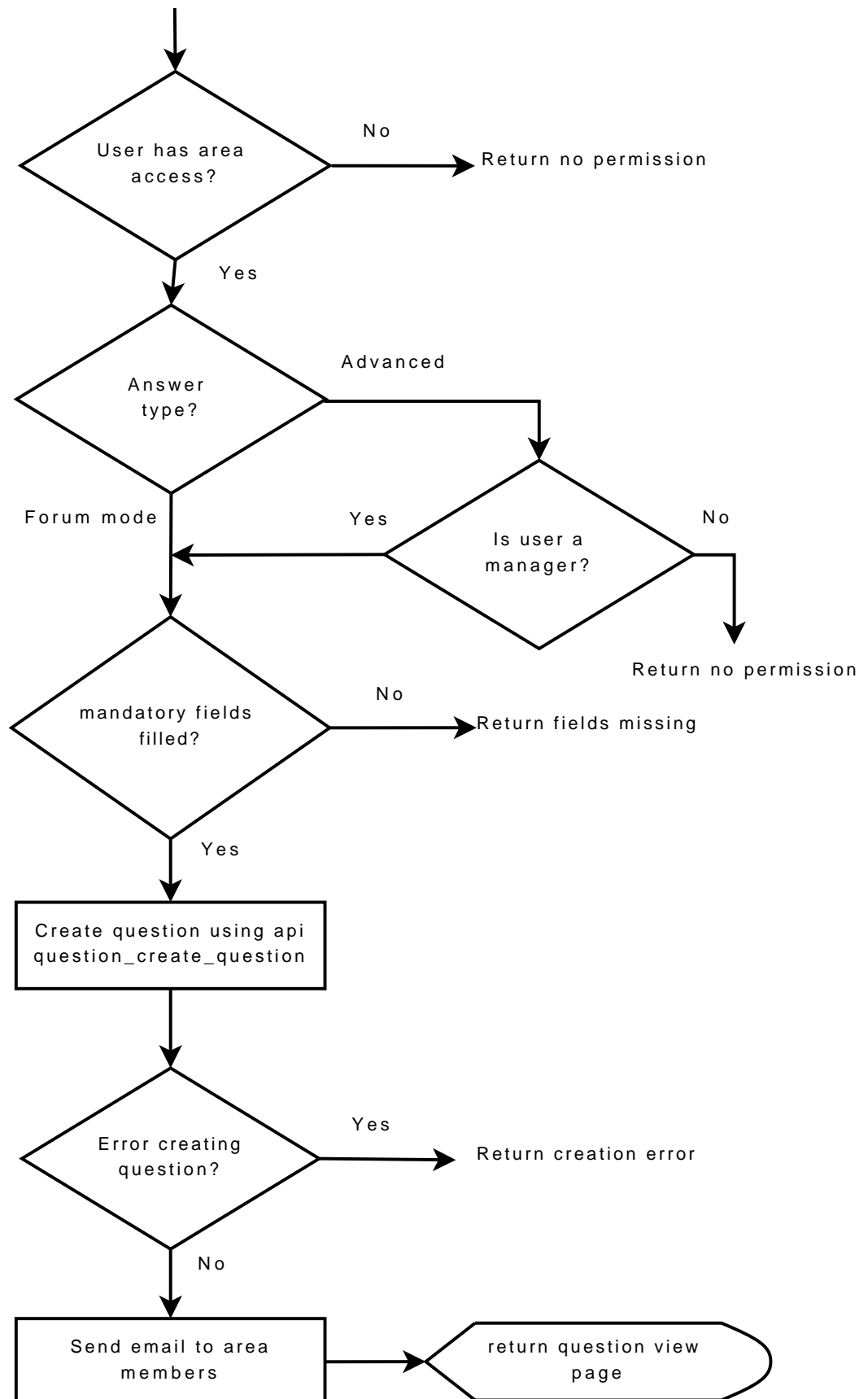


Figure D.2: Create a Question (question submission)

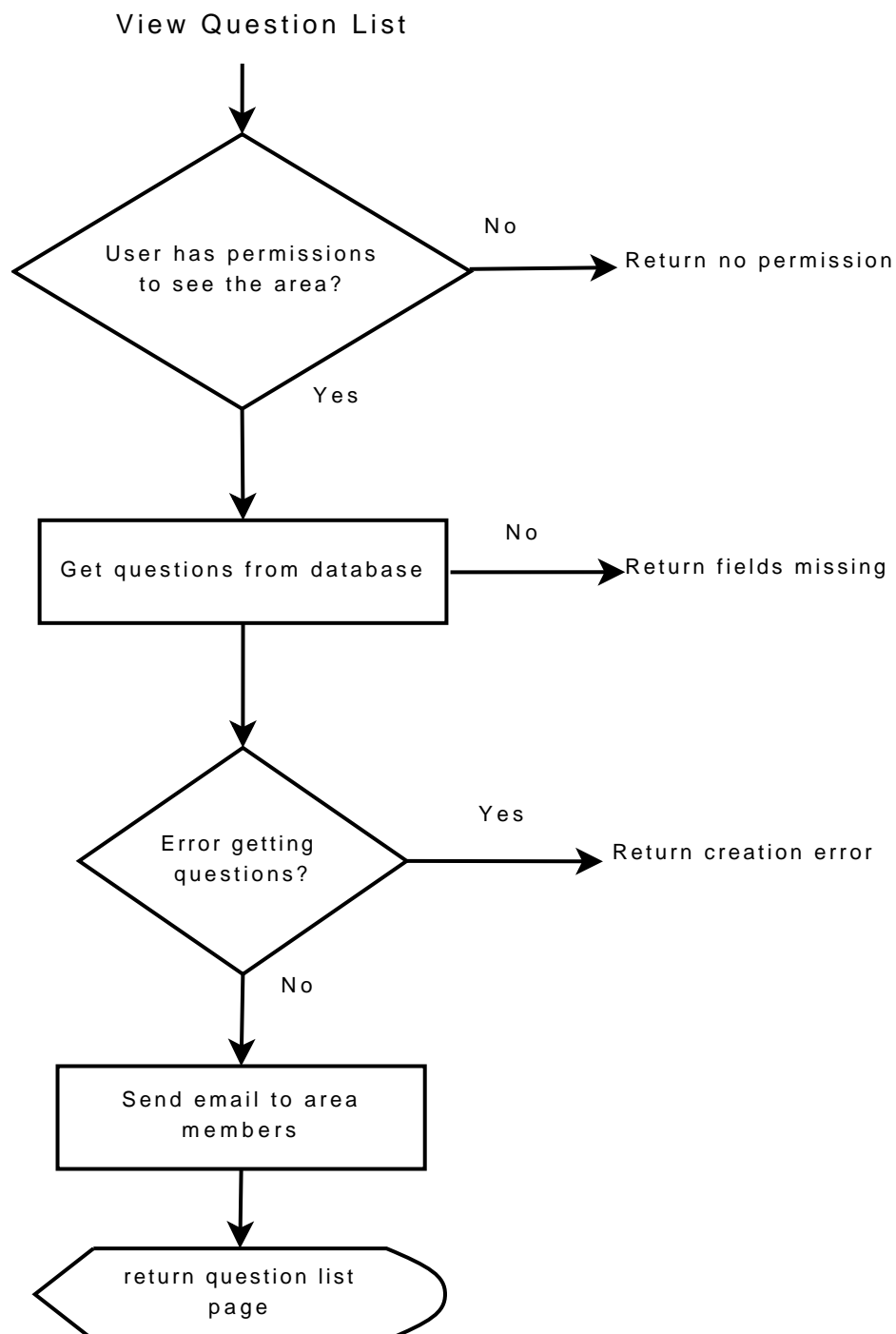


Figure D.3: View Question list

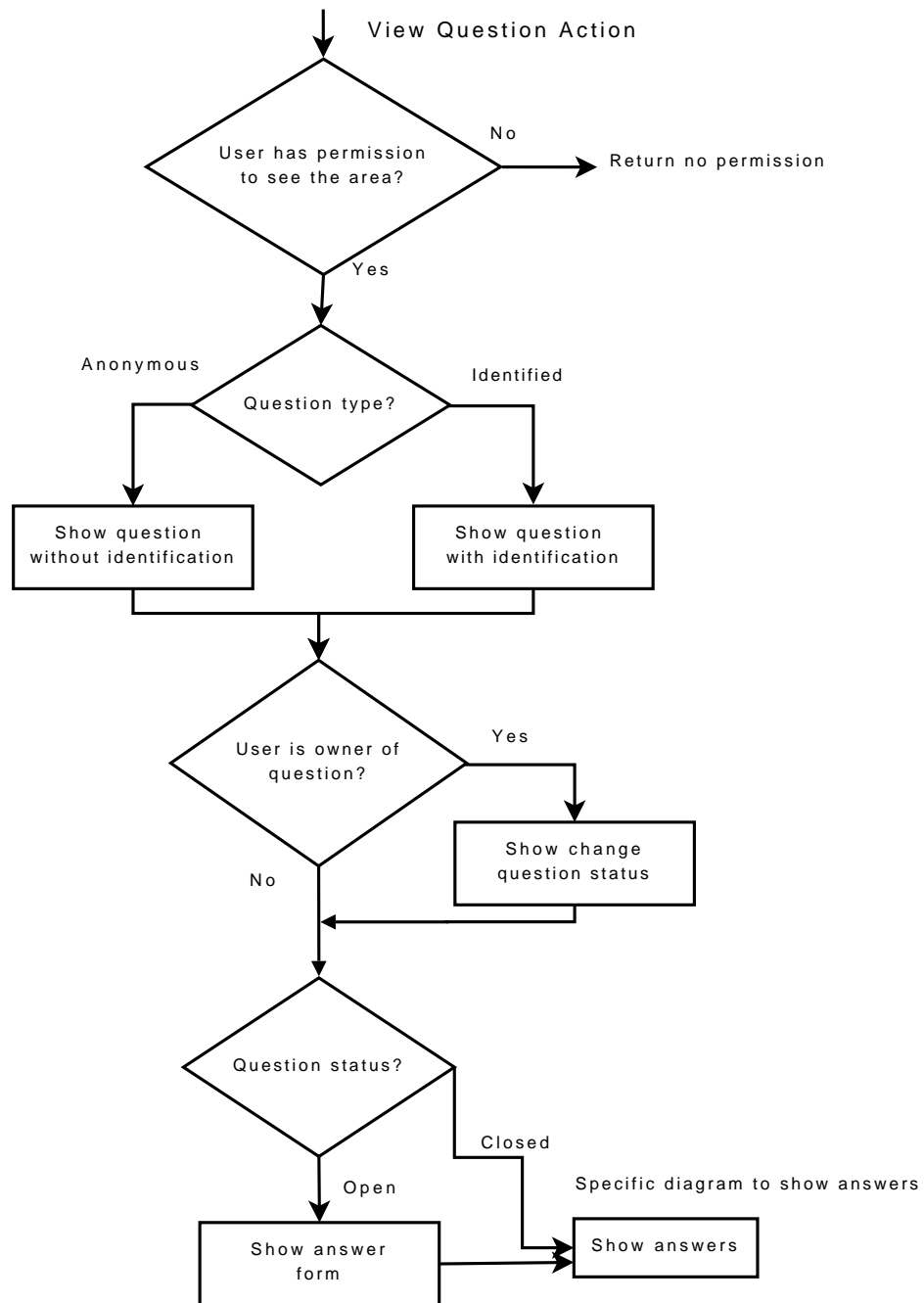


Figure D.4: View Question

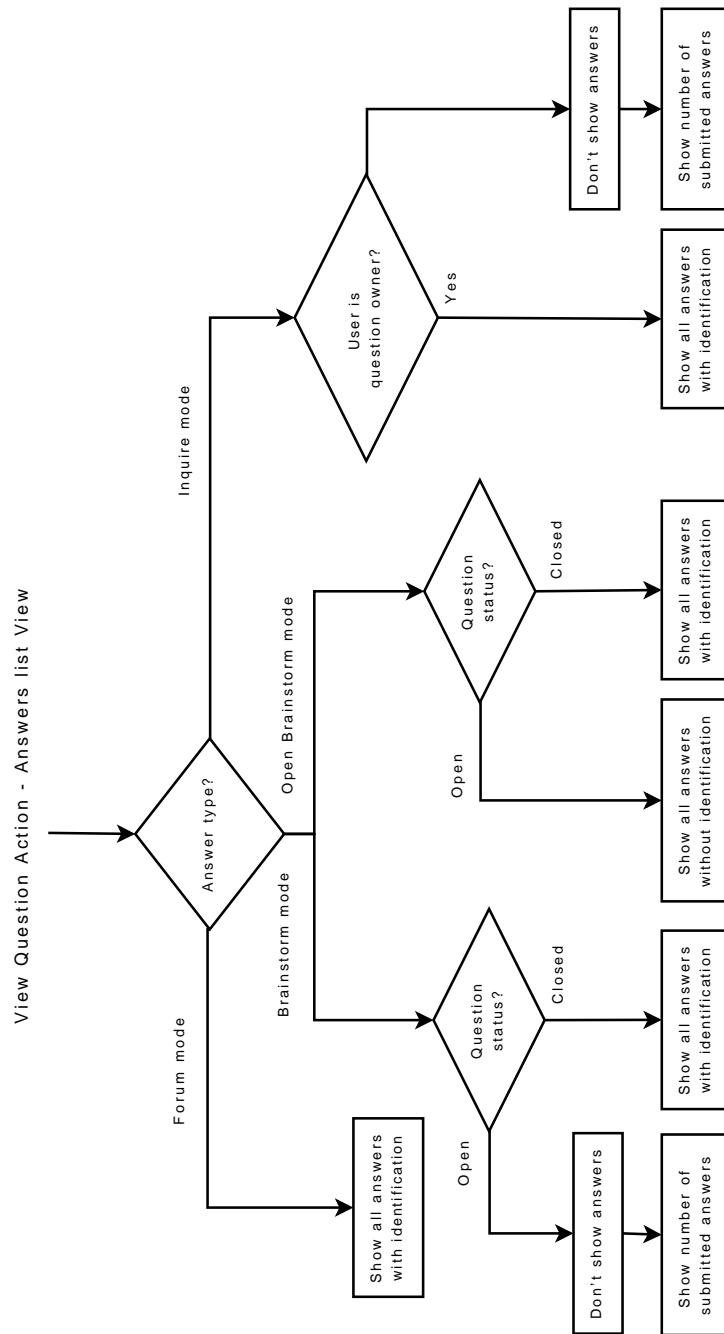


Figure D.5: View Question: Answers list

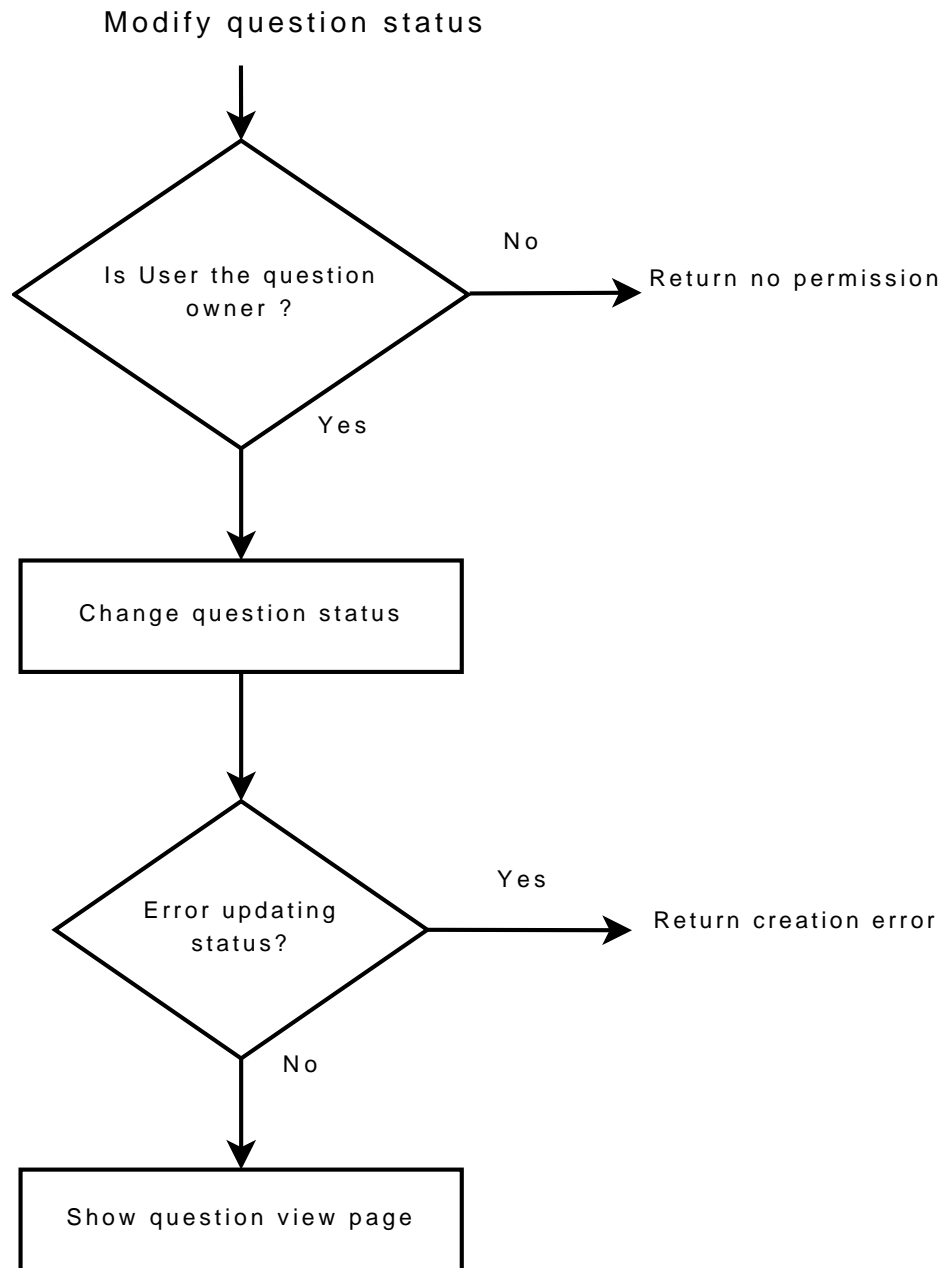


Figure D.6: Modify Question Status

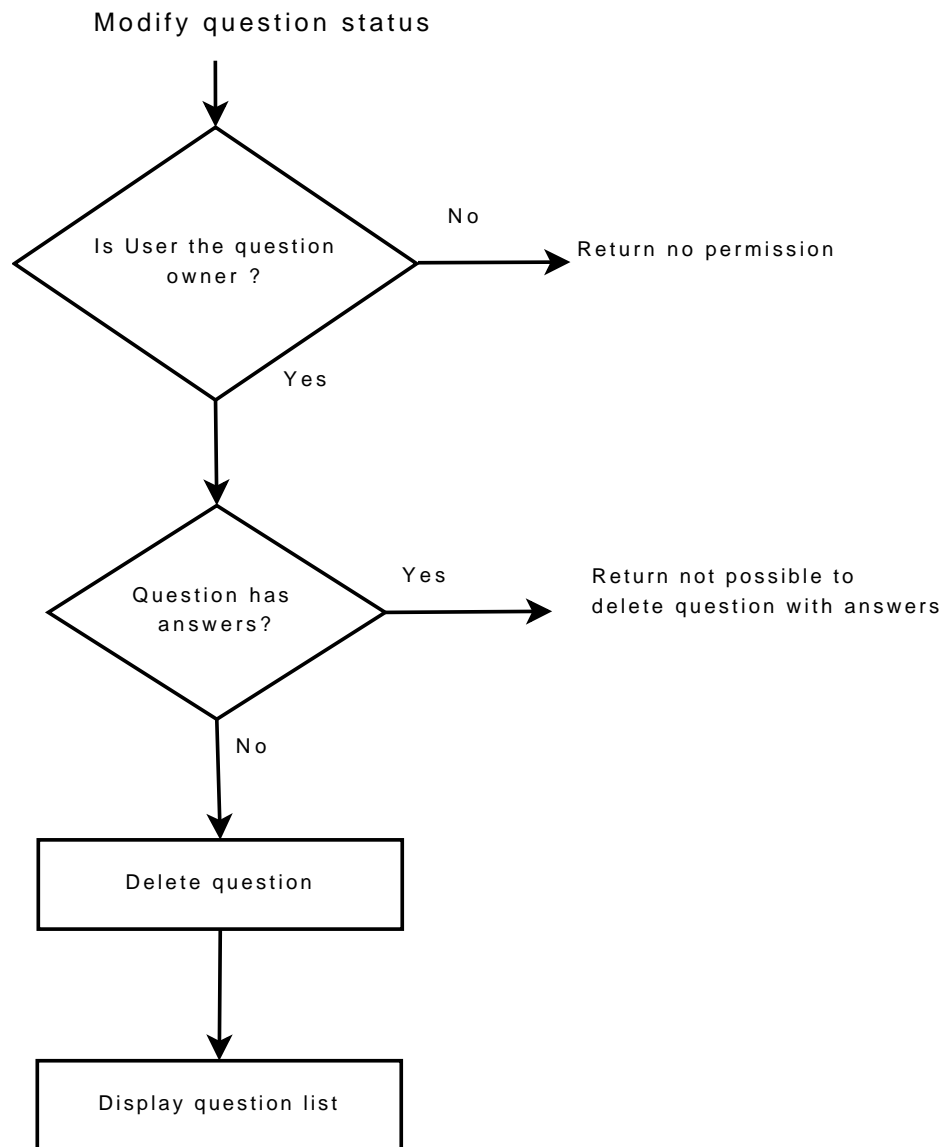


Figure D.7: Delete Question

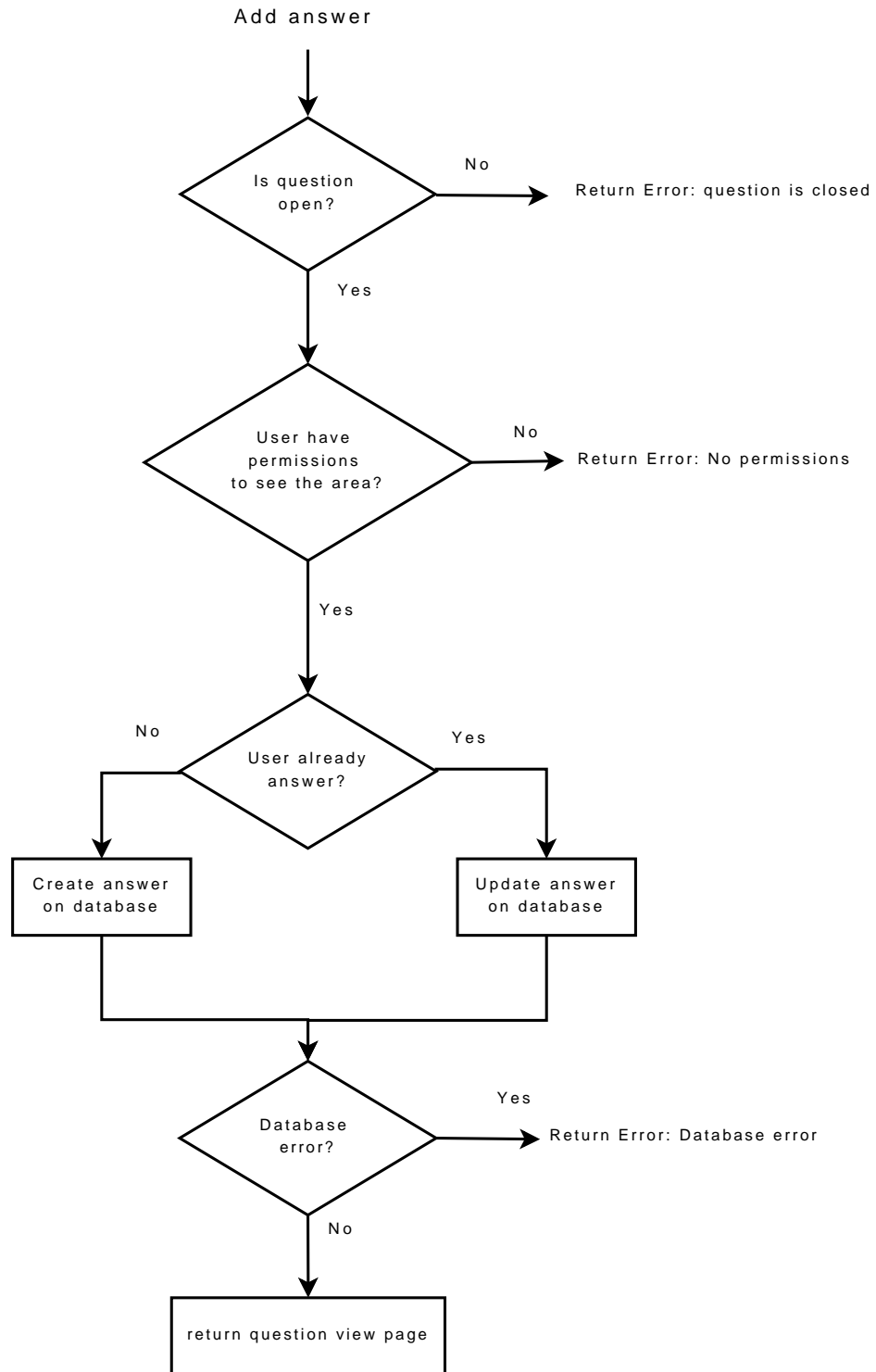


Figure D.8: Add Answer

D.2 Detailed system design: Polls action diagrams

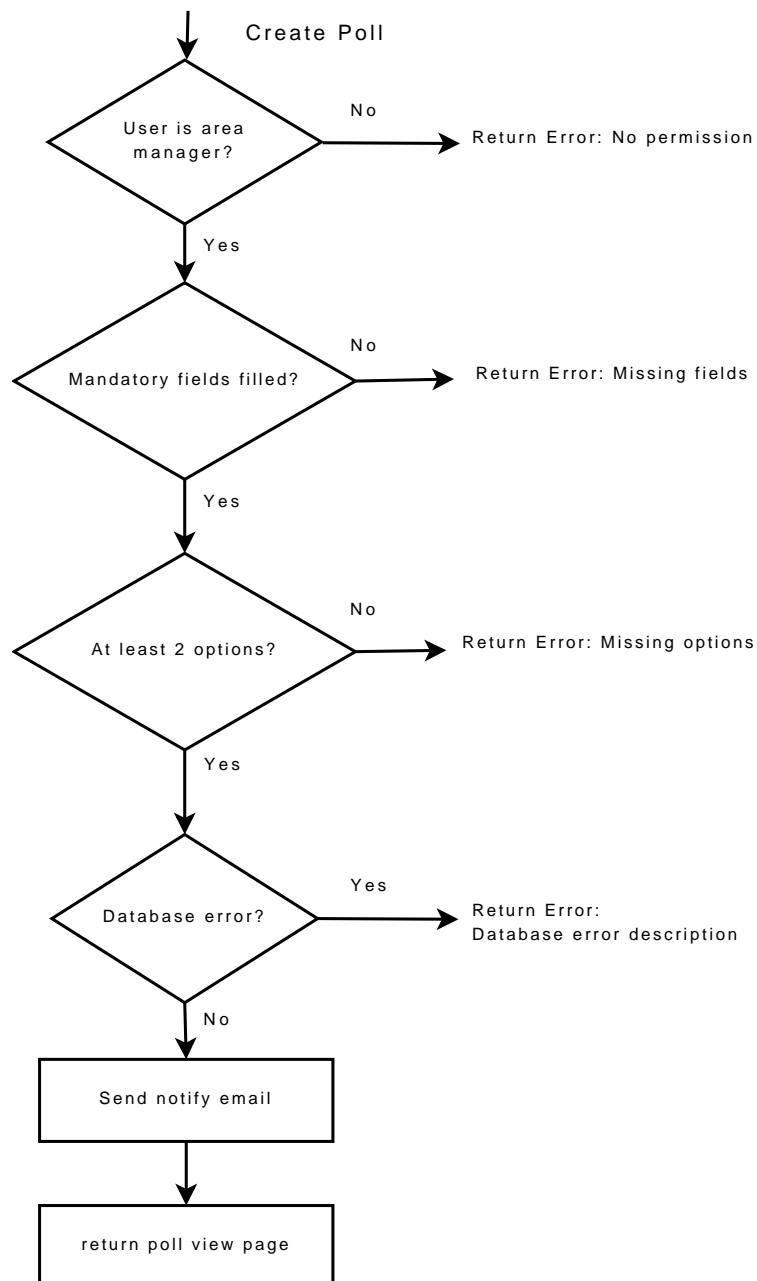


Figure D.9: Create a Poll

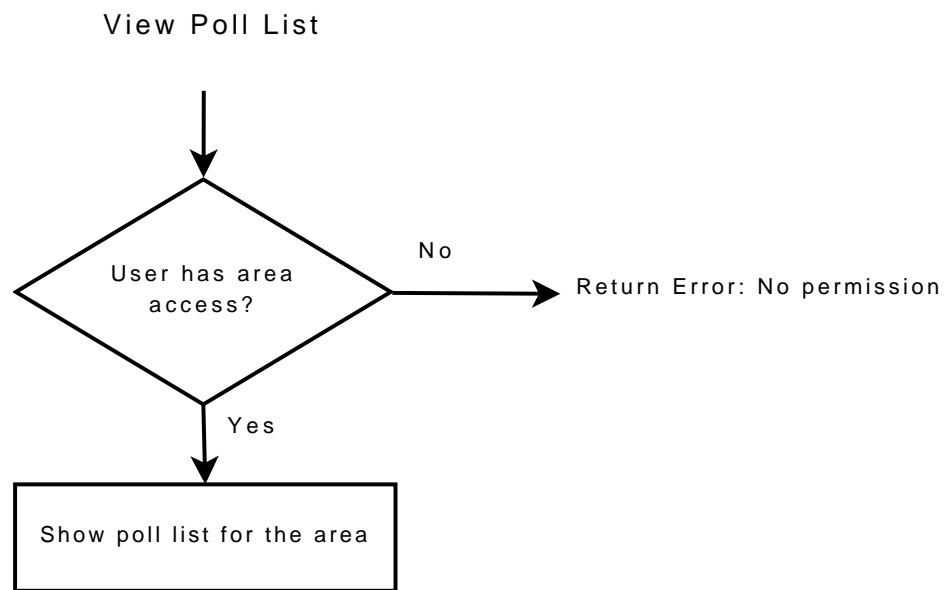


Figure D.10: View Poll list

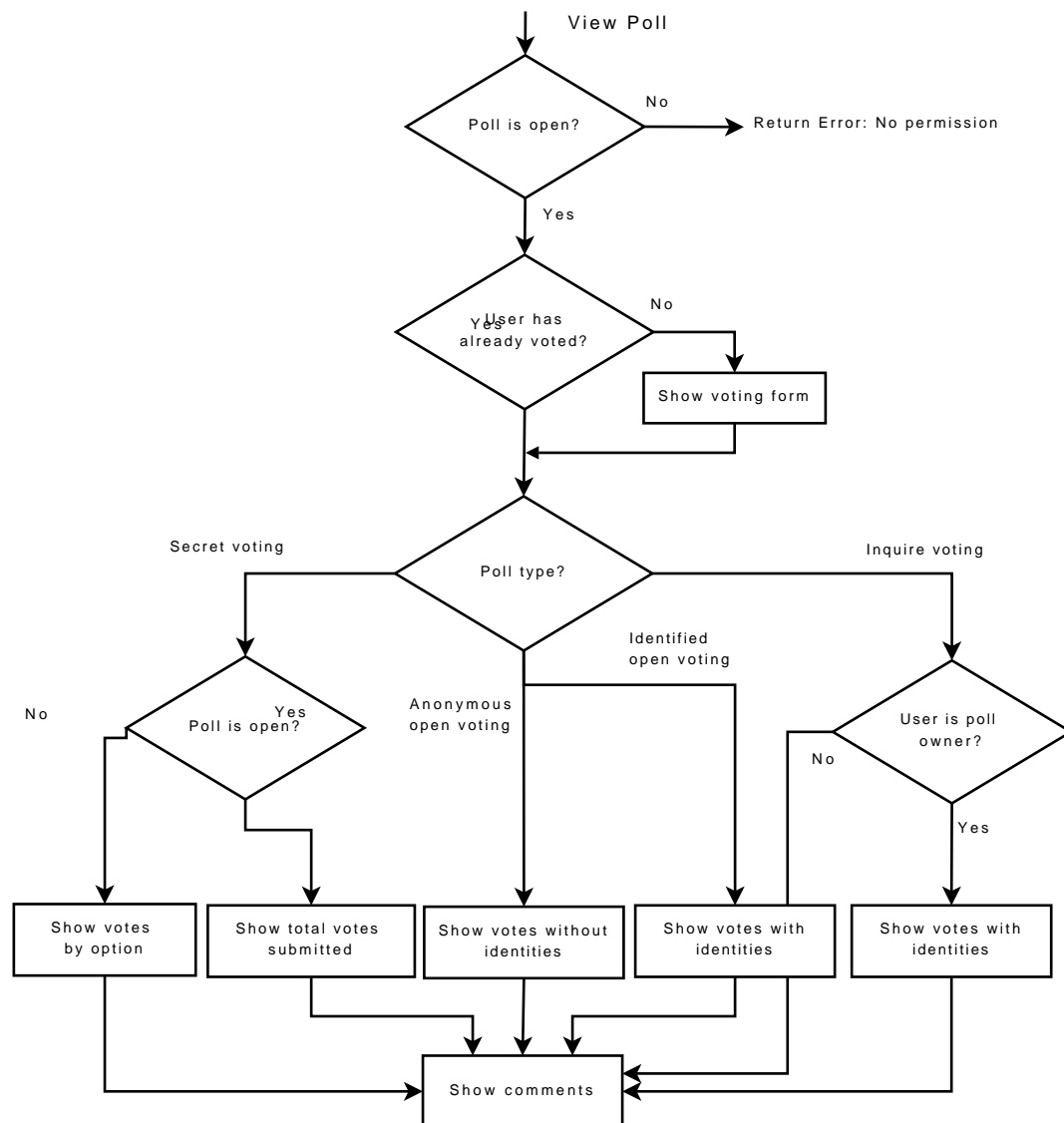


Figure D.11: View Poll

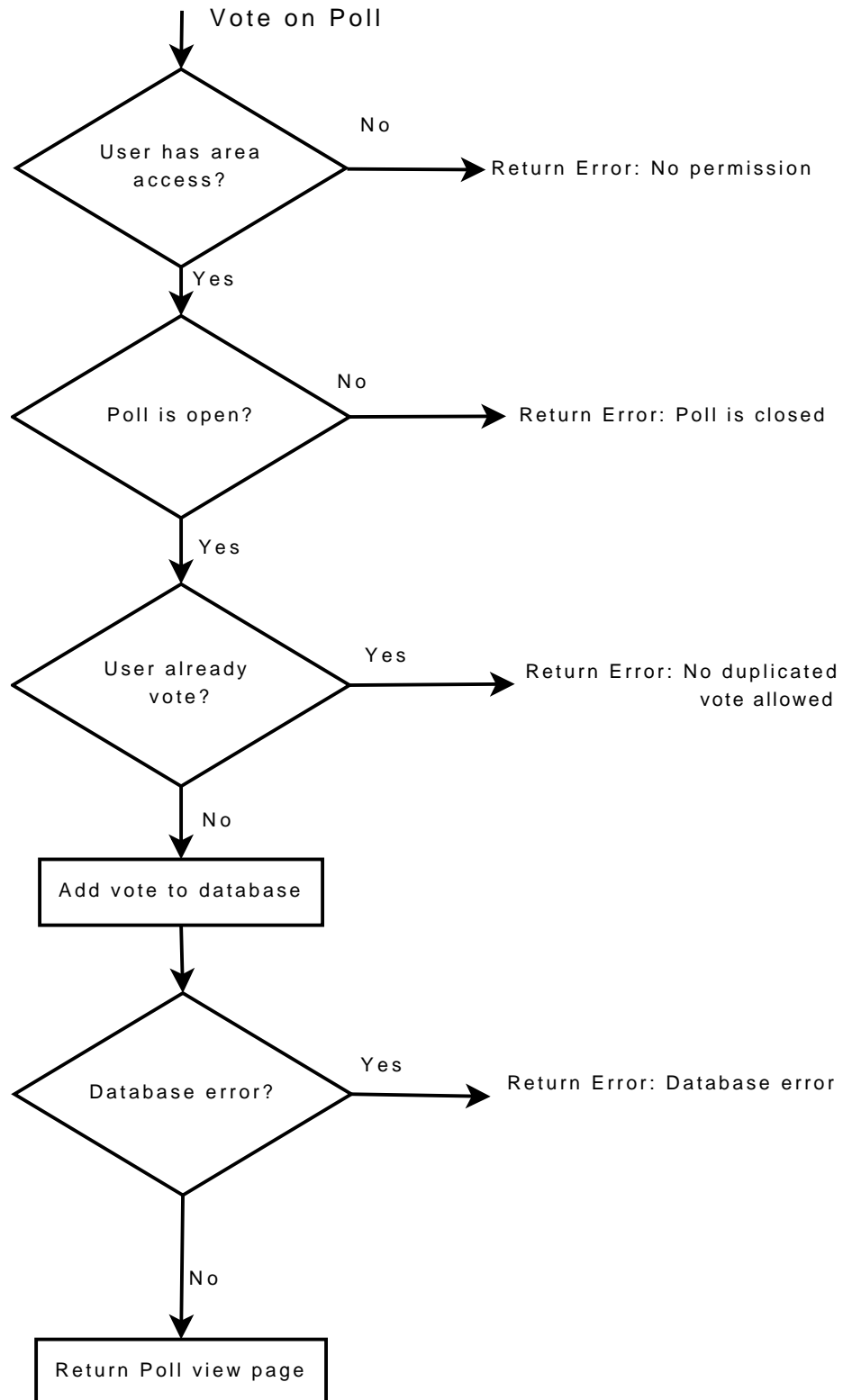


Figure D.12: Vote on Poll

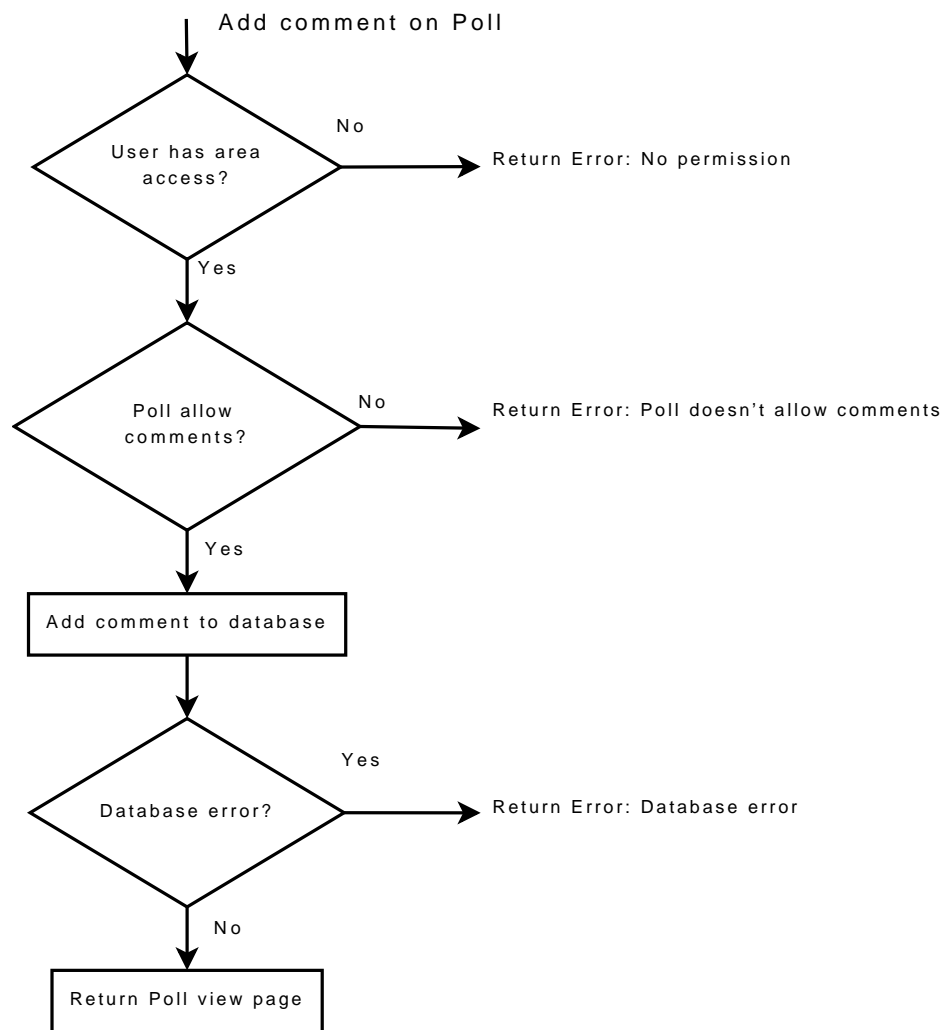


Figure D.13: Add comment to a Poll

Appendix E

PHP Language Overview

E.1 PHP Introduction

PHP (recursive acronym for "PHP: Hypertext Preprocessor") is an open source scripting language used for web development. The run-time engine runs on top of web servers (Apache, Microsoft IIS) and has wide support for common technologies, for example:

- Multiple database support:
 - Adabas D
 - dBase
 - Empress
 - FilePro (read-only)
 - Hyperwave
 - IBM DB2
 - Informix
 - Ingres
 - InterBase
 - FrontBase
 - mSQL
 - Direct MS-SQL
 - MySQL
 - ODBC
 - Oracle (OCI7 and OCI8)
 - Ovrimos

- PostgreSQL
 - SQLite
 - Solid
 - Sybase
 - Velocis
 - Unix dbm
- Output XHTML, XML, text, images, PDF files and even Flash movies
- Support for talking to other services using protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM
- Support for talking to other services using protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM and CORBA
- XML documents processing using the SAX and DOM standards
- Compression utilities (gzip, bz2, zip)
- Authentication Services: KADM5 : Kerberos V and Radius
- Calendar conversions
- Credit Card Processing: MCVE - MCVE (Monetra) Payment and SPPLUS - SPPLUS Payment System
- Cryptography Extensions
 - Cracklib
 - Hash HASH Message Digest Framework
 - Mcrypt
 - Mhash
 - OpenSSL
- Web Services
- SCA
- SOAP
- XML-RPC

The code can be mixed with HTML and used to create output format. in the next code listing is a "Hello World" script in PHP:

```
<html>
<head>
  <title>PHP Test</title>
</head>
<body>
<?php echo '<p>Hello World</p>'; ?>
</body>
</html>
```

The code is separated from the normal html by the strings: `<?php` and `?>`.

The language is simple and powerful and support advanced programming topics, such as function parameters by reference, multidimensional arrays, or even a class concept.

E.2 PHP Variable types

PHP supports eight primitive types. Four scalar types:

- boolean
- integer
- float (floating-point number-double)
- string

Two compound types:

- array
- object (used for class Instantiation)

And two special types:

- resource (resource variables hold special handlers to opened files, database connections and image canvas areas)
- NULL

In the next code listing the variables are initialized to show how to initialize these variable types. Note that in PHP the variables have a `$` prefix.

```
<?php
/* simple types */
$var_bool = TRUE;    // a boolean
$var_str = "foo";    // a string
```

```

$var_int = 12;      // an integer
$var_float = 1.234; // a float

/* array example */
$var_array = array("foo" => "bar", 12 => true);
echo $var_array["foo"]; // print out bar
echo $var_array[12];    // print out 1

/* object example */
class car
{
    function brake()
    {
        echo "Doing brake";
    }
}

$bmw = new car; // $bmw is an object

/* resource examle */
// $connection is a resource for the databas connection
$connection = mysql_connect("localhost", "username", "pass");

?>

```

E.3 PHP Operators

Arithmetic operators used in PHP:

Example	Name	Result
-\$a	Negation	Opposite of \$a.
\$a + \$b	Addition	Sum of \$a and \$b.
\$a - \$b	Subtraction	Difference of \$a and \$b.
\$a * \$b	Multiplication	Product of \$a and \$b.
\$a / \$b	Division	Quotient of \$a and \$b.
\$a % \$b	Modulus	Remainder of \$a divided by \$b

E.4 Assignment Operators

The basic assignment operator is the "=", but it could be combined with arithmetic operators and result for example like:

```
$result += 5; //increment the result variable by 5
```

E.5 Bitwise Operators

Bit operations in a high level language is not expected, but PHP supports it.

The next table show the possible operations.

Example	Name	Result
<code>\$a & \$b</code>	And	Bits that are set in both <code>\$a</code> and <code>\$b</code> are set.
<code>\$a \$b</code>	Or	Bits that are set in either <code>\$a</code> or <code>\$b</code> are set.
<code>\$a ^ \$b</code>	Xor	Bits that are set in <code>\$a</code> or <code>\$b</code> but not both are set.
<code>~\$a</code>	Not	Bits that are set in <code>\$a</code> are not set, and vice versa.
<code>\$a << \$b</code>	Shift left	Shift the bits of <code>ab</code> steps to the left
<code>\$a >> \$b</code>	Shift right	Shift the bits of <code>ab</code> steps to the right

E.6 Comparison Operators

The comparison operators are used to compare 2 values.

Next table shows the operators available to do it.

Example	Name	Result
<code>\$a == \$b</code>	Equal	TRUE if <code>\$a</code> is equal to <code>\$b</code> .
<code>\$a === \$b</code>	Identical	TRUE if <code>\$a</code> is equal to <code>\$b</code> , and they are of the same type.
<code>\$a != \$b</code>	Not equal	TRUE if <code>\$a</code> is not equal to <code>\$b</code> .
<code>\$a <> \$b</code>	Not equal	TRUE if <code>\$a</code> is not equal to <code>\$b</code> .
<code>\$a !== \$b</code>	Not identical	TRUE if <code>\$a</code> is not equal to <code>\$b</code> , or they are not of the same type.
<code>\$a < \$b</code>	Less than	TRUE if <code>\$a</code> is strictly less than <code>\$b</code> .
<code>\$a > \$b</code>	Greater than	TRUE if <code>\$a</code> is strictly greater than <code>\$b</code> .
<code>\$a <= \$b</code>	Less than or equal to	TRUE if <code>\$a</code> is less than or equal to <code>\$b</code> .
<code>\$a >= \$b</code>	Greater than or equal to	TRUE if <code>\$a</code> is greater than or equal to <code>\$b</code> .

E.7 Execution Operator

PHP has a special operator to execute commands in host machine. The operator is the `“`. For example, the follow php script prints out in web page the contents of the server root.

```
<?php
$output = `ls -al /`;
echo "<pre>$output</pre>";
?>
```

E.8 Incrementing/Decrementing Operators

In the same way as C/ C++ the `++` and `--` operators are supported.

E.9 Logical Operators

These operators can be used to conjugate comparisons.

Example	Name	Result
<code>\$a and \$b</code>	And	TRUE if both <code>\$a</code> and <code>\$b</code> are TRUE.
<code>\$a or \$b</code>	Or	TRUE if either <code>\$a</code> or <code>\$b</code> is TRUE.
<code>\$a xor \$b</code>	Xor	TRUE if either <code>\$a</code> or <code>\$b</code> is TRUE, but not both.
<code>! \$a</code>	Not	TRUE if <code>\$a</code> is not TRUE.
<code>\$a && \$b</code>	And	TRUE if both <code>\$a</code> and <code>\$b</code> are TRUE.
<code>\$a \$b</code>	Or	TRUE if either <code>\$a</code> or <code>\$b</code> is TRUE.

E.10 Array Operators

PHP has special operators to deal with arrays.

Example	Name	Result
<code>\$a + \$b</code>	Union	Union of <code>\$a</code> and <code>\$b</code> .
<code>\$a == \$b</code>	Equality	TRUE if <code>\$a</code> and <code>\$b</code> have the same key/value pairs.
<code>\$a === \$b</code>	Identity	TRUE if <code>\$a</code> and <code>\$b</code> have the same key/value pairs in the same order and of the same types.
<code>\$a != \$b</code>	Inequality	TRUE if <code>\$a</code> is not equal to <code>\$b</code> .
<code>\$a <> \$b</code>	Inequality	TRUE if <code>\$a</code> is not equal to <code>\$b</code> .
<code>\$a !== \$b</code>	Non-identity	TRUE if <code>\$a</code> is not identical to <code>\$b</code> .

E.11 PHP Control structures

The PHP have several program control structures:

- `if / else / else if`
Statement for condition control, similar to other languages, c or java.
- `while / do-while/ break / continue`
Loop control with condition to continue declared in while.
- `for / foreach / break / continue`
"for" is used for complex loop control. Usage:

```
for (expr1; expr2; expr3)
    statement
```

The first expression (`expr1`) is evaluated (executed) once unconditionally at the beginning of the loop.

In the beginning of each iteration, `expr2` is evaluated. If it evaluates to `TRUE`, the loop continues and the nested statement(s) are executed. If it evaluates to `FALSE`, the execution of the loop ends. At the end of each iteration, `expr3` is evaluated (executed).

Example:

```
<?php
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
?>
```

The "foreach" is an easy way to iterate over arrays.

Usage:

```
foreach (array_expression as $value)
    statement
foreach (array_expression as $key => $value)
    statement
```

For example consider next example:

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$amp;value) {
    $value = $value * 2;
}
?>
```

- switch

Switch is used to compare an expression with a list of conditions.

Unlike `c++`, strings can be used to compare.

Consider follow example:

```
<?php
switch ($i) {
case "apple":
    echo "i is apple";
    break;
case "bar":
    echo "i is bar";
    break;
case "cake":
    echo "i is cake";
    break;
}
?>
```

- return

This statement is used to terminate immediately a function.

- include / require / require_once / include_once

Used to call a php script on the current script. "require" will cause program to fail if file does not exist, "include" will not cause any failure.

The "include_once" and "require_once" only execute the files if they were not loaded.

These statements are the base to create libraries and organize the code.

E.12 PHP functions

A function is a block of code that can be executed whenever we need it.

See next example of a PHP function code:

```
<?php
function sayHello()
{
    echo "Hello World!";
}

sayHello();
?>
```

E.13 Function arguments

It's possible to pass arguments to be used inside the function.

See next example using arguments.

```
<?php
function add($x,$y)
{
    $total = $x + $y;
    return $total;
}

echo "1 + 16 = " . add(1,16);
?>
```

E.14 Function arguments by reference

You can return one object, but sometimes it is needed that functions return more than one object.

Other usage is when the objects to be copied inside the functions are huge (like images or memory buffers). To answer this problem, PHP uses the

same approach as other languages like c++: pass arguments by reference. The argument passed by reference, can be accessed and modified and those changes will be reflected outside the function.

Example of reference arguments in a function.

```
<?php
function getCustomerInfo( &$customerName, &$customerNumber)
{
    $customerName = "Fernaο Magalhaes";
    $customerNumber = 32;
    return true;
}
?>
```

E.15 PHP classes

Class are an objects abstraction, which can contain variables and methods. The class itself is only executed when some object is instantiated as being the class.

Next example is a PHP class code:

```
<?php
class SimpleClass
{
    // member declaration
    public $var = 'a default value';

    // method declaration
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

In order to use the class some code similar to next code must be used.

```
<?php
$classeInstance1 = new SimpleClass();
$classeInstance2 = new SimpleClass();
$classeInstance1->displayVar();
$classeInstance2->displayVar();
?>
```

Like other languages there are other advanced object oriented features supported by PHP:

- Constructors and Destructors

- class visibility using keywords: public, protected or private
- static members or methods
- class constants
- Class Abstraction: defining abstract methods to be implemented by classes extending the abstract class
- Interface classes: class definitions for classes implementing the class interface
- Overloading: ability to add methods to a class in run time.
- Reflexion: get class interface information in runtime.

Bibliography

Alter, S. (1975). *A Study of Computer Aided Decision Making in Organizations*. PhD thesis, MIT.

Ferguson, R. L. and Jones, C. H. (1969). "A Computer Aided Decision System". *Management Science*, 15, pp. 550–562.

Fischer, G. (1999). "A Group Has No Head - Conceptual Frameworks and Systems for Supporting Social Interaction". *Joho Shori*, 40, pp. 575–582.

Gerrity, T. P. J. (1971). "Design of man-machine decision systems: an application to portfolio management". *Sloan Management Review*, 14, pp. 59–75.

Gerstberger, P. G. and Allen, T. J. (1968). "Criteria Used by Research and Development Engineers in the Selection of an Information Source". *Journal of Applied Psychology*, 4(52), pp. 272–279.

Gray, P. (1981). "The SMU decision room project". In *Transactions of the 1st International Conference on Decision Support Systems*, pages 122–129, Atlanta, USA.

Hackathorn, R. and Keen, P. (1981). "Organizational Strategies for Personal Computing in Decision Support Systems". *MIS Quarterly*, 5, pp. 21–26.

Heylighen, F. (1999). "Collective Intelligence and its Implementation on the Web: Algorithms to Develop a Collective Mental Map". *Comput. Math. Organ. Theory*, 5(3), pp. 253–280.

Huber, G. P. (1982). "Group decision support systems as aids in the use of structured group management techniques". In *Transactions of the 2nd International Conference on Decision Support Systems*, pages 96–103.

Iansiti, M. and Richards, G. L. (2006). "The Business of Free Software: Enterprise Incentives, Investment, and Motivation in the Open Source Community". *Working Paper Series, Harvard Business School*, No 07-028.

- Licklider, J. C. R. (1992). "Man-Computer Symbiosis". *IEEE Ann. Hist. Comput.*, 14(1), pp. 24.
- MacCormack, A. (2007). "Innovation through Global Collaboration: A New Source of Competitive Advantage". <http://www.hbs.edu/research/pdf/07-079.pdf>.
- Morrison, E. W. and Milliken, F. J. (2000). "Organizational Silence: A Barrier to Change and Development in a Pluralistic World". *The Academy of Management Review*, 25(4), pp. 706–725.
- Morton, M. S. S. and Stephens, J. A. (1968). "The impact of interactive visual display systems on the management planning process". *IFIP Congress*, 52, pp. 1178–1184.
- Morton, S. (1971). *Management Decision Systems: Computer-based support for decision making*. Division of Research, Graduate School of Business Administration, Harvard University, Boston, USA.
- Paulus, P. B. and Nijstad, B. A. (2003). *Group creativity innovation through collaboration*. Oxford University Press, New York, USA.
- Peck, M. S. (1987). *The Different Drum: Community Making and Peace*. Simon & Schuster, New York, USA.
- Power, D. J. (2002). *Decision support systems: concepts and resources for managers*. Quorum Books, Westport, Conn.
- Power, D. J. (2007). "A Brief History of Decision Support Systems".
- Rodriguez, M. (2005). "Advances towards a General-Purpose Societal-Scale Human-Collective Problem-Solving Engine". <http://arxiv.org/pdf/cs/0501004>, 0501004.
- Rodriguez, M. A. and Steinbock, D. J. (2004). "A Social Network for Societal-Scale Decision-Making Systems". <http://arxiv.org/abs/cs/0412047>, s0412047.
- Rodriguez, M. A.; Steinbock, D. J.; Watkins, J. H.; Gershenson, C.; Bollen, J.; Grey, V.; and deGraf, B. (2007). "Smartocracy: Social Networks for Collective Decision Making". In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 90, Washington, DC, USA. IEEE Computer Society.
- Simon, H. A. (1976). *Administrative Behavior*. The Free Press, New York, USA.

Taleb, N. N. (2007). *The Black Swan: The Impact of the Highly Improbable*. Random House, New York, USA.

Turoff, M. and Hiltz, S. (1982). "Computer support for group versus individual decisions". *IEEE Transactions on Communications*, COM-30:1, pp. 82–90.

Turoff, M.; Hiltz, S.; Cho, H.-K.; Li, Z.; and Wang, Y. (2002). "Social Decision Support Systems (SDSS)". In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 1*, page 11, Washington, DC, USA. IEEE Computer Society.

Wikipedia (2008). "Free and open source software". http://en.wikipedia.org/wiki/Free_and_open_source_software. [Online; accessed 3 Nov 2008].