**G52GRP**

**Democratic Conferencing Tool**

**Interim Report – November 2009**

**Group:** gp09-sdb

**Supervisor:** Dr. Steven Benford

**Group Members:** Robert Golding, William Redrup**,** Tammie Seo**,** Christopher Lensvelt**,** Henry James**,** Zhongda Zhu

Table of Contents

[Table of Contents 2](#_Toc247097335)

[1 The Problem 3](#_Toc247097336)

[2 Background Information & Research 4](#_Toc247097337)

[2.1 Survey of Existing Systems 4](#_Toc247097338)

[2.2 Market Research 4](#_Toc247097339)

[2.3 Technical Research, Platforms & Tools 4](#_Toc247097340)

[2.3.1 Source Code Management 4](#_Toc247097341)

[2.3.2 Project Hosting 7](#_Toc247097342)

[2.3.3 Collaboration 8](#_Toc247097343)

[3 Requirements Specification 9](#_Toc247097344)

[3.1 Functional Requirements 9](#_Toc247097345)

[3.2 Non-Functional Requirements 10](#_Toc247097346)

[4 Initial Design & UI 11](#_Toc247097347)

[5 The Platform/Implementation 12](#_Toc247097348)

[5.1 Web-Based 12](#_Toc247097349)

[5.2 Desktop-Based 12](#_Toc247097350)

[6 Initial Implementation/Prototyping 13](#_Toc247097351)

[6.1 Prototype Specification 13](#_Toc247097352)

[7 Problems Encountered 13](#_Toc247097353)

[8 Time Plan 13](#_Toc247097354)

G52GRP Interim Report

# The Problem

Project Description: *“This project is to design, implement and evaluate a new text conferencing tool that aims to support more democratic participation. Each participant is given a "talk budget" which may be refreshed from time to time. When this runs out they can no longer talk, unless another participant donates some of their remaining budget to them.”*

To clarify, our group project is to design and build a Democratic Conferencing Tool. Our primary aim when building this application is to introduce a new product to a market brimming with competition. To combat this however, we should be introducing completely new ideas and implementing them into our product while also focusing on the features that make the existing competition so popular. For this to work, it is important to carry out adequate research into every area of our project, ranging from platform suitability to utilising the best method to communicate within the group.

The key feature that was identified immediately was that this product must differ from a conventional chat tool. Simply because the primary focus is that rather than just offering a place for people to chat about random topics without structure, our application should offer users a system allowing them to join existing debates where they can share their own structured opinion on any of the subjects being discussed, or allow them to begin their own subject and open it for discussion. Either way, the primary goal is to provide a place to go for those who want to take part in more serious discussion than you find in a regular web chat room.

Secondly, another key aspect of the system is that it must be democratic, as indicated by its name. To ensure this aim remains true is vital for us as it will help set us apart from the competition mentioned earlier in the report. To ensure this is the case a feedback system should be implemented involving votes by the users within a conference room, allowing good or bad feedback to affect whether the current speaker should continue speaking freely regarding their opinion, or whether the amount they say about a subject should be limited if they receive all bad feedback.

Again, to ensure this product is noticeably democratic compared to competitors within the market, something to consider would be allowing users the freedom to moderate amongst themselves, rather than implementing a set of administrators that impose themselves upon the rest of the user-base, which would be considerably more like a dictatorship and this is obviously the opposite of one of our key aims within the project.

In essence, what we hope to build is a democratic, user-friendly product where opinion-sharing is not only welcome, but encouraged. Whilst heading towards this objective, the biggest challenge we have encountered thus far is collaborating as a group whilst trying to implement key decisions. One of the first things we found it necessary to narrow down would be the set of system rules during our low-tech analysis stage. Unfortunately, this process was far more involved than we originally anticipated, and caused us to question our decision numerous times both while meeting informally and formally as a group. The reason this caused such indecision amongst us as group members is that we had differing ideals about various aspects of the system and how it should be implemented. However, eventually a decision was made to narrow down the rule-sets and we

# Background Information & Research

## Survey of Existing Systems

We have carried out research into similar systems, in order to gather a better understanding of features that are considered ‘a must’ for this type of project, and those that could perhaps be improved upon. Two particular projects that caught the attention of our researcher were Google Wave, which unfortunately is currently limited to invitation testing only (could be considered a closed beta), and Effusia Business Messenger, a very popular tool for conferencing in the business sector. More on this can be found in the research notes.

## Market Research

## Technical Research, Platforms & Tools

### Frameworks

We carried out some initial research into the suitability of different platforms for developing our application. Broadly, there were two areas in which we could continue: web-based platforms and desktop-based platforms. The group looked into a number of different options regarding each type of platform. For example, an application that extends a social networking website (such as Facebook) could be used. This would mean that a large part of the system would already be present (such as the user sign-up and login/authentication modules).

#### Web Frameworks

If we decide to develop a web-based system, then we could either code the entire system from scratch, or use an existing framework. Frameworks can make development much faster, and could save the group a lot of time – allowing us to implement more advanced features into the final system. However, poorly designed frameworks can inhibit the development effort – if the API is difficult to work with.

##### Django

Django is a Python-based web MVC (Model-View-Controller), developed in a newsroom environment where speed is of the essence. It has an elegant API, and includes a template engine and a database ORM – which is compatible with most common database formats (including SQLite, MySQL, PostGres and MS SQL).



Models in Django are represented as simple classes, with attributes of the class representing the database fields. Foreign key and many-to-many relations are simple to describe. Here is an example of a simple model in Django source code:

class Poll(models.Model):

question = models.CharField(max\_length=200)

pub\_date = models.DateTimeField('Date published')

class Choice(models.Model):

poll = models.ForeignKey(Poll)

choice = models.CharField(max\_length=200)

votes = models.IntegerField()

These model classes define two database tables (objects) – Poll objects and Choice objects. This is the heart of a basic Django application for voting on polls.

Django uses a simple Python function for the view, which is called from a particular URL location (for example, **/users/rob/**. Here’s an example of a simple view in Django:

def index(request):

latest\_poll\_list = Poll.objects.all().order\_by('-pub\_date')[:5]

data = {'latest\_poll\_list': latest\_poll\_list}

return render\_to\_response('index.html', data)

This view uses Django’s ORM to fetch a list of the latest five poll objects from the database, and sends that data to a template named **index.html**.

Templates in Django are very simple, and are basically written as HTML, with some basic logic and variable replacement. For example:

{% if latest\_poll\_list %}

<ul>

{% for poll in latest\_poll\_list %}

<li>{{ poll.question }}</li>

{% endfor %}

</ul>

{% else %}

<p>No polls are available.</p>

{% endif %}

Our “lead software engineer” – Rob – has extensive experience working with Django, and has developed a number of projects using this framework in the past.

##### Ruby-on-Rails

#### Desktop-Based Platforms

### Source Code Management

We felt that version/source control would be essential to the smooth operation of our group project. We needed a fast and simple way to share code and documentation – whilst not treading on each other’s feet whilst working on different parts of the project.

Using a Source Code Management system allows multiple group members to work on the same file or document at once. When editing binary files, however, (such as Microsoft Office documents), the SCM is not able to “diff” the file - that is, to see the differences between the two. This means that it is left to the user to either overwrite the old file completely, or throw away all their changes when a collision occurs.

This shouldn’t be a problem for our group, however, as a single group member is responsible for collating all sections of, and submitting, the documents required in our deliverables. That is, one group member will be actually committing the documents to the repository - though other group members may work on them in a separate fashion.

To this end, we conducted some research into Version Control Systems. The results of this research are included below.

#### Subversion

Subversion (or SVN) is a **client-server** based Source Control Management tool. That is, the repository resides on a central machine (the server) and each user checks out a “working copy” (the client). Each time the user makes a set of changes, they commit them to the server. Commits are stored in a linear fashion, one after the other. In this way, SVN manages the version (or **revision**) of each file using a number, that increments by one on every commit. This allows the user to instantly see which version of a file is the latest – by comparing revision numbers.

Each document is stored as a hard-links to the most recent revision of said file, keeping the overall size of the repository down.



#### Mercurial

Mercurial is a **distributed** Source Control Management tool. It differs from Subversion in that there is no central machine on which the repository is stored. Instead, each user has their own copy of the repository, which they commit to locally when their changes are complete. To share changes, users synchronise their local repositories.

This means that the structure of a Mercurial repository is not linear, as with Subversion. Instead, it could be described as a **Directed, Acyclic Graph**. The diagram below shows a representation of this graph.



The diagram above shows the following situation; the repository is copied from state **C1** into state **C2** and **C2’,** by two separate users. These users then make their changes, and commit to their local copes - which results in states **C3** and **C3’** respectively. These two states can then be **merged**, resulting in a hybrid of the two previous states.

The non-linear nature of a Mercurial repository means that there are no revision numbers as in Subversion. Instead, each revision is identified by an **SHA-1 Hash** of the contents in that revision. The hashes in a Mercurial repository look like this:

1:1ef7872431f9c64908c732f0bcd4db5700b4cb70

This means that unlike Subversion, two revision identifiers cannot be easily compared to see which is newer.

### Project Hosting

It was obvious early on in our research that the group would need a centralised location to store our code and documentation, which would be accessible from anywhere. That is, we needed a place to host our project’s repository.

#### Google Code

GoogleGoogle provides a free online project hosting service, called **Google Code**.

It offers either a Subversion or Mercurial repository included as standard, though it requires that the code for the project be made open-source. Also, it includes a Wiki section - with all pages contained in the repository - so anything stored on the Wiki is easily backed up by checking out or cloning the repository.

Google code requires all users to have a Google account.

#### SourceForge

SourceForge is a free web-based project hosting site, run by GeekNet.

It offers a huge array of services, including a shared repository, SSH access, Wiki, forums, and downloads. It is, however, a difficult and length process to set up a new project.

### Collaboration

Also, the group would need a way to collaborate whilst working on the project - as we would not all be in the same room at all times.

#### Basecamp

Basecamp is an online service offering group collaboration and project management, developed by a company called 37signals.



Basecamp is free for small projects, and allows users to track deadlines and to-do lists, post messages, and chat in real-time. Also, project milestones can be managed using the suite, which can be very helpful when working against the clock.

Perhaps most importantly, Basecamp offers rich email integration. When a message is posted, every member in the group is notified of that message – and can reply via email. This is then interpreted by the Basecamp software, and posted automatically onto the project’s page. This not only ensures that group members don’t miss an important announcement, but also saves considerable time logging into the web application to post a reply.

#### Project2Manage

Similarly to Basecamp, Project2Manage is an online project-management suite.



It is free for one project, though this does limit the feature set somewhat – as file sharing and time-tracking is not permitted in the free plan. This is assumed to mean that neither deadlines or milestones can be tracked using Project2Manage unless a paid plan is purchased – unlike Basecamp which offers these features in its free plan.

Alongside SVN, we have also implemented Google Code into our plan for file sharing. In short, Google code offers users access to various developer tools and products. Google also has Project Hosting, which has enabled us to create a directory for our group project to store all documentation and code that is created. As this doesn’t allow for updates to be shared instantly with each member while working, it will only be used to share documentation upon completion, whereas SVN can be used for all ‘in-between’ updates.

On Google Code, we currently have a directory specifically for sharing files with the other group members, a ‘Wiki’ for typing and sharing documents such as the meeting minutes with each other, which comes with a date sub-heading allowing us to differentiate between the different meeting minutes, and the additions/changes made to them since.

Something else that is important for our project is possibly utilizing Google App Engine, which Google uses for hosting web applications created by its users. As we have decided upon creating our project as a web application, rather than desktop-based, this is an important feature provided by Google for us.

Also, the group chose to utilize the free Subversion hosting offered by Google, in the form of **Google Code**. Using this service meant that the group had a central location to store code and documentation, that would always be available and, perhaps more importantly, accessible through a web interface. This gives members the ability to browse the code repository without actually checking out a copy, which is very useful for quick status-updates.

# Functional Specification

## Overview

The project (named DemoConf until a final name is chosen) is a service that allows users to participate in a democratic debate or discussion - to achieve a goal of agreement on a particular subject.

The system forces users to vote on a goal at the end of each time period (which is 10 minutes by default). The goal (usually an issue that must be debated) is specified when the conference room is created. If all (or a specified proportion) of the participants agree when the vote is cast, then the goal has been achieved and the conference can be ended. Otherwise, the discussion continues for another period.

The graphics and layout of the system shown in this specification are merely to illustrate the underlying functionality. The actual look and feel will of the final product will be developed over time.

## User Stories/Scenarios

The user stories or scenarios specified in this specification will help to define how users interact with the system.

### Scenario 1: John

John is a senior developer on the open-source Ubuntu Linux project. He often needs to discuss important issues regarding the project with other developers, who are located in different countries around the world. At present, this is achieved by using a combination of IRC, and AIM (Internet Relay Chat, and AOL Instant Messenger) to chat online.

Acting on a recommendation from his friend, John signs up for an account with **DemoConf** - the online debate and discussion service. The next time John has to discuss an issue with this peers, he directs them all to the service also, and sets up a conference room. The team join the room and start to discuss.

The team find that the system encourages decision-making, and they reach a decision after 3 periods of 10-minutes. Afterwards, John receives an email from Philip, one of the development team, saying how well the discussion went - and that he feels they should use **DemoConf** every time they want to discuss an issue.

## Flowchart



## Functional Requirements

1. The system should have a web-accessible interface
2. The system should allow users to login and logout
3. The system should present users with a list of available conferences
4. The system should allow users to enter a conference, and chat with others in that room in near-real time. This requires the user to be logged-in.
5. The system should allow users to see who else is in the conference
6. The system should allow users to leave a conference, and return to the list of available rooms
7. The conference system should use an asynchronous method of communication (i.e. the page should not refresh when a new message is available.
8. The system should allow users to sign-up for a user account.
   1. Users must choose a username and password
   2. Users must enter their email address
   3. The system should send the user an email, which includes a link that the user must click in order to “activate” their account
   4. Before the user has clicked the link in their email, the account should be inactive - so the user cannot login
9. Users should be able to change their password.
10. Users should be able to change their email address.
11. Users should have a profile, containing their personal information shared with other users.
12. Users should be able to edit the information in their profile.
13. Users should be able to create conferences.
14. When a conference is created, it should not be published immediately - instead residing in a private area until the user chooses to “publish” it and allow other users to join.
15. Users should be able to edit conferences in their private area, before they are “published”.
16. Users should be able to delete conferences in their private are, before they are “published”.
17. Once a conference is “published”, it should not be editable or removable.
18. The system should require users to input a “poll” when creating a conference. This should represent the “objective” of the conference - all members should agree on the issue named in the poll.
19. The system should allow users to choose the length of the “period” when creating a conference (a default value of 10 minutes should be provided).
20. The system should switch to a vote at the end of each specified time period.
21. When in the voting mode, users should not be able to enter messages in the conference. Everyone is required to vote on the poll specified at creation time.
    1. If all members in the conference agree, then the conference can be officially called finished - and the objective achieved (this should be decided by a vote).
    2. If all the members do not agree, another period is entered to try and resolve the disagreement. At the end of each period, the voting mode is entered again.

## Non-Functional Requirements

1. The web interface should be accessible from any platform, using either of four main browsers - Firefox, Google Chrome, Internet Explorer and Opera
2. The system should be secure in that unauthenticated users cannot access conferences
3. The system should be available
4. The system should respond to requests in a reasonable time period

## User Interface

The designs below show the potential layout for the conference room screen - as this is the most customised screen in the system. Both implement most of, but not the entire entire feature set of the system.





# The Platform/Implementation

Firstly, the group had to decide whether to develop our application in one of two ways: a web-based application, or a desktop-based application.

## Web-Based

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| Faster development | Limited to one-way communication (request-response) |
| User-interface layout simplified (HTML + CSS) | Interactive UI is more difficult (requiring JavaScript or Flash) |
| Does not require installation of a client |  |
| Truly multi-platform |  |
| Possibility of SaaS (Software as a Service) |  |

## Desktop-Based

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| Two-way communication possible (via sockets) | Slower development (custom protocol may be required) |
| Implementing an interactive UI is easier | User-interface implementation is more difficult |
|  | Requires installation on all client machines |
|  | Hard to make truly multi-platform application |
|  | SaaS is ruled-out |

The group felt that the advantages of a web-based application are so great that our development should focus on this style.

Though the fact that a web-based application is forced to adhere to the request-response nature of HTTP may require some creative programming, we feel that this is an obstacle that can be overcome.

Next, the group needed to decide on a language and/or framework to use for the development of the project. Broadly, we had two options at this point - we could either write the entire application from scratch, or use a web framework to ease the development and speed up the process. However, frameworks can hamper development if they are too inflexible - so this choice is vital.

There are a number of web frameworks that would be suitable for a project such as this - Ruby on Rails, CakePHP, and Django to name but a few. However, Rob has prior experience programming using the **Django** web framework, which is written using the Python programming language. Though there was a belief that Django would be suitable for our project – the only way to know for sure was to build a prototype, to act as a “feasibility study” – in order to find out whether we could indeed implement our project using Django. The results of this study are recorded in the next section of this report.

# Initial Implementation/Prototyping

## Prototype

In order to find out whether the Django web framework would be suitable for our project, the group decided that we should build a prototype system. It was decided that this system should include only the core functionality of the final system - as a sort of feasibility study. This is because it was still not known whether the request-response nature of the web (and Django itself) would be entirely suitable for this kind of system. To this end, we came up with a very simple specification for the prototype.

This is simply a subset of the list of things that the final system **should** do. The reason for this is because the prototype is intended to find out whether Django is going to present any major problems for our project, and whether the actual system design is actually progressing in the right direction.

### Prototype Specification

1. The system should have a web-accessible interface
2. The system should allow users to login and logout
3. The system should present users with a list of available conferences
4. The system should allow users to enter a conference, and chat with others in that room in near-real time. This requires the user to be logged-in.
5. The system should allow users to see who else is in the conference
6. The system should allow users to leave a conference, and return to the list of available rooms
7. The conference system should use an asynchronous method of communication (i.e. the page should not refresh when a new message is available.

The last point above (point 7) is important - as it ensures that the system does not simply refresh the web page in order to check for new messages. Though this would work, it was deemed totally impractical, as it would interrupt the user every time the refresh occurred, not to mention the added load that would be placed on the web server. This would be the real test as to whether the framework has the functionality (and the group has the skills) to develop such a system using web-based technologies.

### Prototype Results

The results of the prototyping are summarised below. Some screenshots of the system are included, to show the initial workflow.







The prototype was successfully implemented using the Django framework, and fulfilled all aspects of the specification. The result of this prototype is more than just a simple yes/no answer - as to whether Django is suitable - rather it allowed us to find out about any potential issues we may encounter while working with Django.

# Problems Encountered

# Time Plan

Still to be added:

More recent UI designs, information about play-testing session, research notes, summary of progress to date.