Technical Report for the Volunteer Management System

Team Lunchbox
Bardh Jahjaga
Brian Lee
Evan Williams
Martin Marks

Project Sponsor
Jennifer King

Faculty Coach
Kenn Martinez
Table of Contents

Project Overview
Basic Requirements
Constraints
Development Process
Project Schedule: Planned and Actual
System Design
  Quality Attributes that drove the Architecture
  Architecture
Process and Product Metrics
Product State at Time of Delivery
Project Reflection
References
Project Overview

The Volunteer Management System being created by Team Lunchbox is designed to facilitate and simplify the process of distributing information about volunteering opportunities and acquiring volunteers for said opportunities to a specific group of individuals. This will be accomplished through an individualized website that will allow users to post and sign up for volunteering opportunities specific to the organization hosting the deployment of the site they are using. The intent is to speed the creation and acquisition of volunteers for organizations. It is also intended to provide data on the volunteers to the organization. Other goals include mitigating the impact of budget cuts on organizations, strengthening community ties and freeing up staff from the laborious process of matching jobs to volunteers.

For further information and understanding, please refer to ‘Brockport Volunteer System Synopsis’ and ‘Project Plan,’ documents 1 and 2 in the references.
Basic Requirements

The site will accommodate three types of users: Posters, Helpers, Volunteer Coordinators. Posters are pre-approved community figures that are in search of volunteers for a certain need. Helpers are community-approved volunteers who are looking to contribute volunteer hours of service or donations. Helpers will be able to search and sign up for volunteer or donation requests, view current and past requests, view and complete evaluations, and print a report detailing their volunteer history. Posters will have all the capabilities of Helpers, but will also be able post jobs. Volunteer Coordinators are administrators who are aware of the communities volunteering rules and any legal criteria for posted needs; they have the capabilities of Posters and are also responsible for approving or rejecting need requests, as well as creating/approving user accounts. A Volunteer Coordinator can also view volunteering data for an organization.

The system will account for user error by allowing most actions to be undone: Posters can cancel their need requests and resubmit rejected requests, Helpers can cancel their attendance to need requests that they have signed up for, and at any time Volunteer Coordinators can change the level of users (or ban them outright).

The system also provides notifications to users via email when: a user’s account is approved or rejected, a need request is approved or rejected, an evaluation is now available or has been completed, as well as reminder emails prior to the start of a job.

For further information and understanding, please refer to ‘SRS - Volunteer Management System,’ document 3 in the references.

Constraints

Due to this being the creation of the project, not a follow up to an existing project, and the
rather flexible nature of the sponsor, there are virtually no constraints placed on the team. The project needed to be usable by anyone ten years old or older, and had to reduce the time required to create a volunteering need. Creating a user interface that a ten year old can use is largely dependent on grammar used, and therefore is not overly complicated to achieve. As for reducing the time to create a volunteering need, the current method that the project needed to beat was so wildly variable, that no accurate estimate of a time to beat could be obtained by the sponsor. Other restraints included that the project must be free to use, and free to implement, as well as be a website.

For further information and understanding, please refer to ‘Project Plan’ and ‘SRS - Volunteer Management System,’ documents 2 and 3 in the references.
Development Process

The development process the team used was the Scrum methodology, with slight alterations to better fit a team that was not in a traditional workplace environment. The team performed 2 week sprints, and would have stand-up meetings each time they met, at least three times a week, though sometimes as often as five times a week. The team met with their sponsor once every two weeks, though earlier on when the project was still being defined, they would meet with her every week, and in later stages, would send her e-mail updates when she was not in attendance. The sponsor approved of the process and changes. She pointed out her unfamiliarity with the area of software engineering process, and deferred to the team’s judgement, counting on them to make the best decision for the project’s success. There were four main roles, including Technical Lead, Project Manager, Usability Expert, and Testing Lead. The Technical Lead handled selection of technologies and tools, set up of the virtual machine, team website, SVN repository, and other tasks of a similar nature. The Project Manager handled creating agendas, reviewing sprints, encouraging team members to inform each other of progress and barriers, communicating progress to the sponsor, gathering metrics and data and interpreting this information, and several other tasks. The Usability Expert researched usability as applied to interfacing with a wide array of users, both in terms of age, location, and technical background, and planned the usability testing performed by the team. The Testing Lead created test plans for the various stages of testing, as well as code review templates, and set up automated testing and testing tools.

Tracking of tasks was performed by using Acunote, which showed each team member’s tasks for the current sprint. This afforded the team a great deal of visibility into the progress being made on each task. It was also convenient that the team’s bug tracking tool, Mantis, integrated with Acunote to allow for bugs to be tracked as tasks.
For further information and understanding, please refer to ‘Project Plan,’ document 2 in the references.

**Project Schedule: Planned and Actual**

The team came up with a schedule based upon the limitations of the class, i.e., finishing in May, and the time already used planning and gathering requirements. Based upon the two week sprints decided upon for the project, a goal was set of May 8th to finish development. In addition to this, due to the importance of usability to the project, it was decided that the user facing aspects of the project should be the initial focus, to aid in both presenting progress to the sponsor, as well as allow usability testing to occur earlier in the project. Originally there was a plan to perform usability testing around the spring break, hoping to take advantage of the extra time.

Our actual schedule did not go quite as planned. The development of the project’s views took more time than was expected. Also, it took the team longer to get its development efforts up to speed. Once development was moving at a reasonable pace, the goal of usability testing by the break had passed, so it was delayed, and not performed until late March. This put the team a few weeks behind. However, by focusing on development work and not allowing any teammate fall behind on their work, the team was able to make up for lost time. The goal of completing the project by May 8th is no longer realistic, however, the team has finished the critical functionalities prior to the end of the Spring quarter.

For further information and understanding, please refer to ‘Project Plan,’ document 2 in the references.
System Design

Quality Attributes that drove the Architecture

The three main quality attributes that had to be the focus of our architecture: Usability, Security, and Maintainability.

Usability is important because many of the reasons for the creation of the site involved lack of usability in the current method. In the design, this was supported by planning to keep the code that was user facing separated from the rest of the system, so should either side need to be altered, the opposing side would be able to remain intact.

Security was immediately evident as an important aspect to the system, due to the initial deployment involving children as a main user group. Security would be important to the system regardless; however, with children, it required an even higher degree of concern. As a result, there were many layers implemented to protect the system from intrusions and attacks.

Users are authenticated with a username and password to access the system, and can only access areas of the system appropriate to their user level. The only sensitive piece of data that we store is the users password; furthermore, we store it using a very secure encryption mechanism. In terms of data access, all database interaction is isolated from the client and cannot be accessed from the outside, and all submitted data is sanitized and filtered prior to processing. There is also an intrusion prevention system developed, that logs cross-side scripting and intrusion attempts, blocks the user from accessing the site, and informs the Volunteer Coordinator, however, the logging ability was was deemed unnecessary, so at the moment, is disabled.

Maintainability was deemed important because no matter how much the team was able to get done with the project, the sponsor would very likely think of something after deployment, or be informed of something by the users once the system is deployed, that would improve the
system. Even if this did not happen, due to the idea of opening this system to any organization that desires it, the likelihood that they may want to add or change something is also very high. To assist future development being smooth and as simple as possible, the team came up with a few solutions. First, there was a plan to comment the code to encourage understanding in future users, so that they could comprehend what was done without having to dig in to the work, which could deter their interest. The team’s efforts were also documented, recording what was done, how it was done, plans, and how plans changed to accomplish the team’s efforts. Finally, the team followed a MVC architectural pattern, so that future developers could work on additional parts to the system with fewer concerns of breaking the existing parts of the system. This is evident as we have the ability to turn on/off complete system features through true/false values in the configuration file.

**Architecture**

Our main architecture is a layered architecture, with each layer representing a different set of responsibilities. The idea behind the layered architecture was to make our system as customizable and reusable as possible.
The top layer is our client-side presentation layer. In this layer we have our client side views. These views represent the styling of our system. The reason for this separation is to add another level of usability and modifiability. We want to be able to change the styling at any time without affecting the form and presentation of the system. On top of the styling, we also have UI functionality such as: charts to display user data, expandable forms, date-pickers, etc. Once again, due to the separation of UI styling vs. UI form/presentation, we are able to easily change the look and feel without affecting what we’re presenting.

The next layer is our server-side presentation layer. In this layer we are using the CodeIgniter MVC framework. Instead of starting from scratch, we chose to use something that exists, supports our needs and has a good reputation for such systems; therefore, CodeIgniter was a perfect match.

The first system in the server-side presentation layer is the CodeIgniter Security System. The CodeIgniter Security System filters all input for any cross-site-scripting attempts. All input is filtered prior to being used by any other part of the system.

Then we have our server-side views. Our server side views include our templates, common views and component views. Reusable views and templates allow us to reuse code and easily change many views at once. Component views allow specific parts of the system to be easily changed without affecting the rest of the system. Also, since the server-side views are only concerned with the presentation and the form of the system, we have isolated style changes to the client-side views: thus, increasing the level of usability and maintainability.

Lastly for the server-side presentation layer, we have the controllers. The controllers are the gateway of interaction between the user and the system. They call on the models to retrieve data and appropriately pass that data back to the views.

Our next layer is the application layer. The application layer consist of the models which are
responsible for all application logic. The models talk to the persistence layer through the CodeIgniter database libraries in order to retrieve data. We separate the SQL from the Models into a hidden data access layer that is provided through the CodeIgniter active record libraries. We wanted our application to be database agnostic: therefore, if we ever decide to migrate to a new DBMS we would be able to do this without affecting our application logic. This furthers our support for modifiability/maintainability.

Lastly, we have the persistence layer. This layer contains our DBMS and the actual file disk. This DBMS is responsible for all our data access and management.

When we added the ability to send out reminder emails at configurable intervals, we had to set up some system that would live outside of the web-service and would be able to achieve this. This reminder services uses Java to access our persistence layer and retrieve necessary data to send out the reminder emails.
Process and Product Metrics

The team gathered data for a number of metrics, including velocity, which is standard for any team using a scrum methodology. We also captured feature hours per sprint to see spikes or lag in development. The team also recorded metrics relating to QA, including: bugs/issues found per sprint to show a possible lack of quality in development work done, and bugs resolved per sprint to defend the quality of work. The velocity was broken down into a few different visible results including: work estimated vs work remaining to show how far off the estimates were, effort breakdown by sprint to show where the team’s efforts were concentrated, and individual estimate vs work remaining to show if any individual was falling behind. This helped the team keep on top of whether they were overlooking things, or focusing too little on important areas. It also helped the team see if they were overreaching or under-performing.

The metrics at first showed a hesitation to work on development tasks, which were largely being left until the end of the sprint, and then never completed. However, this, along with an agreement as a team to focus on development, which was needed in order to see the project completed, largely improved this, leading to a massive jump in development by all team members, and visible productivity that the sponsor was very grateful for.

The results of the feature hours completed were fairly predictable. They began with a small jump in sprint 3, followed by a lull, which was largely due to the interim presentations and the between quarter break. This was followed by increasing productivity through sprint 7, a lull again in sprint 8, which is likely due to further presenting. After this there was a decrease from sprint 7 in sprint 9, due to a large shift of focus to testing. To match this, there was a large increase in bugs found sprint 7, the sprint with the most development work, a large decrease for sprint 8, matching the decrease in development, followed by a resurgence of bugs found in sprint 9. Many of the bugs discovered in sprint 7 were solved in the same sprint, which helped verify the validity and quality of the work performed by the team. Also, in sprint 8, almost twice
as many bugs were resolved as were reported. In sprint 9, the bugs resolved and reported were
almost equal. All of this helps defend the quality of the product the team has developed.

As far as velocity is concerned, it was somewhat of a confusing metric. In general, there were
few instances where a teammate was far off from their estimate, though it did happen. Also,
the largest gaps from estimates often happened on the less intense sprints. However, by an
large, the estimates did get better as time went on. The only disagreement with this statement
is in sprint 9, which can be accounted for by teammates highly overestimating the amount of
work able to be completed in an attempt to finish any remaining development work for the
project.

For further information and understanding, please refer to ‘Project Plan’ and ‘Work/Effort
Metrics,’ documents 2 and 5 in the references.
Product State at Time of Delivery

At the time of writing this document, the project is nearing a state of functional completion. Almost all requirements initially discussed have made it into the system. Of the remaining requirements not likely to be included, there are a few main ones, including reposting a need from a Poster’s history, and setting up a need to occur multiple times, or according to a schedule (weekly, monthly, etc). These were left out mainly due to them being convenience features, non critical options that, while possibly making a user’s experience better, are not necessary in the site. Another small feature not included was skills for a volunteering request. The team deemed this unnecessary, as they could be listed in the description. Also, while it was originally planned to allow a user to attempt to resubmit a rejected need, this was left until the end, and turned out to be more involved than expected. Again, this would be a convenient function, but not having it does not damage the ability of the system to perform its goals. Additionally, sorting results by anything other than volunteering or donation, as well as the order results are displayed in, was not implemented. Filtering results was found to be much more complicated than the value it delivered, and ordering results seemed useless once the system began to come together.

Despite these missing points in the project, the key functionality is all there. Users can register, sign up for, withdraw from, create, cancel, and browse needs. Volunteer Coordinators can approve and reject users. Users and needs can be evaluated. In addition to the requirements, there were a few additional features that either were overlooked, or are so common, were not thought of. These include: editing details of a user’s profile, requesting lost login information and a messaging system contained within the site.

An issue the team encountered was the great deal of effort required to developing a usable and refined user interface. The team went for a smooth, easy user experience, that allowed
tasks to be completed quickly, and the site to be understood easily. They also did their best to have a uniform look and feel to the site, as well as be pleasant to view and navigate. After multiple rounds of usability testing, and having a design consultant review the site, the team is confident that, like all software, their work could be improved, that it is a highly usable and easily understandable system.
Project Reflection

After working together for over six months, the team has truly come together, knowing each others’ strengths and weaknesses, as well as where certain members will thrive and do their best work. Also, the team has greatly improved on their ability to estimate the effort needed for tasks, as well as be able to support each other with tasks a teammate is struggling with, seeing that a teammate is struggling before they are overwhelmed. The team did a great job of adapting the process they decided to use to their own specific needs, so that they were not simply going through the motions when performing process related tasks. They also did a good job at adapting to different needs of the projects at different times, moving from documenting and planning to developing to testing, to reviewing without having to spend excess time getting up to speed.

However, with all teams, some things could have gone better. There was some concern that the team could have made a better choice of process. While an agile process was clearly beneficial, Scrum required far more of the team than they could realistically give. This called for them to adapt it to their team, which, while acceptable to a degree, can also be a sign that it is not the process you need. Another issue encountered was the time needed to set up tools, technologies, and development environments on the VM and the teams’ personal computers. The team also underestimated how much training would be required for less experienced team members in order to make a meaningful contribution. It may have been better to chose technologies that the team was more familiar with, or to start earlier when attempting to learn and use the technologies. It would also have been helpful to have reviewed the fundamentals of our MVC framework, which could have encouraged better development sooner. Additionally, the team spent far too long discussing possible features that the system could have. The essential requirements should have been nailed down early, with design and development starting shortly after; this would have allowed the process to account for additional and changing requirements. Instead, the team spent weeks reviewing and planning and drawing
out requirements and use cases. While this gave the team a strong starting point, it did not give them the best standing at the midway point of the project.

Another issue the team encountered was a lack of maintenance of their documents. While there were few changes from creation to delivery in the documents, these changes should have occurred when the change occurred in the system, as opposed to months later. This falls primarily on the Project Manager, and perhaps on a lack of communication between the team when things changed. Also, in a similar vein, many of the images describing the design of the project were abandoned shortly after their creation and inclusion in a document. This again falls on the Project Manager for not seeing that they need to be updated, and partially on the team for not making all images available in an accessible and editable format.

Finally, testing should have been performed from the very beginning of development. Instead, the team did not have a testing structure set up until mid to late in the second quarter and test cases were started a bit after that. While the team did manage to create a number of tests that did uncover some bugs in the system, this shouldn’t have been left until the tail end of the project. It highly conflicts with the ideals of their chosen process, which brings back the question of “was the correct process chosen?”.

References

1. Brockport Volunteer System Synopsis
2. Project Plan
3. SRS - Volunteer Management System
4. VMS - Architecture
5. Work/Effort Metrics