Museum Experience Survey

Jonathon Shippling
Andrew Mueller

Edward Dinki
Robert Vrooman

Lockheed Martin
Sponsors: Paul Mittan, Tom Monaco, Donna Jones-Wright, Cheryl Dutko

Faculty Coach
Professor Larry Kiser
Table of Comments

1. Project Overview
2. Basic Requirements
3. Constraints
4. Development Process
5. Project Schedule: Planned and Actual
6. System Design
7. Process and Product Metrics
8. Product State at Time of Delivery
9. Project Reflection
10. References
Project Overview

Lockheed Martin has a number of volunteer members involved with The Discovery Center of the Southern Tier, a non-profit museum, where Lockheed Martin has sponsored the addition of a number of new exhibits highlighting engineering. In order to improve the Discovery Center, Lockheed Martin wishes to implement a system for tracking user engagement and get feedback on the exhibits. The purpose of this project is to create that system.

The Museum Experience Survey will provide an electronic system for visitors of the museum to provide feedback and demographic statistics with as little manual entry as possible. The Museum Experience Survey will ask visitors basic demographic questions such as the ages of the children visiting, whether or not it’s a first-time visit, as well as allow the visitor to rate and provide feedback on the exhibits. Volunteers working at the museum will be able to see the data and statistics received from the Museum Experience Survey and use that data to better the museum.

There are only three users of the system: child visitors, adult visitors, and admins. Child visitors are generally in the 4-12 year range, but vary slightly from this. The children will take the survey on a windows tablet via the chrome browser in the lobby of the museum. The adults will have a similar survey environment, but the tablets will be placed in the gift shop of the museum. Admins will be volunteers at the museum who have the responsibility to create new exhibit questionnaires and view data, statistics, graphs, and more information collected from the system.
Basic Requirements

Survey

The key feature of this application is that it allows users to take surveys. Therefore the first and most important requirement is that the system supports users taking survey and saving the responses. The surveys must support a variety of question types, which include multiple choice with a single answer, multiple choice with multiple answers, slider questions, short answer questions, and a question type that allows users to pick their favorite exhibits.

Targeted Design

The system shall support targeted design for both adult and children user groups. The Discovery Center desires a version of the survey that is more child friendly so that children are more likely to take and complete the survey. While not imposing too many strict requirements, the staff did say they want at least brighter colors and more graphics in the children targeted design. The target design requirement was the lowest priority requirement going into the project but made it into the final product in the form of themes.

Survey Configuration

The application is required to support the configuration of surveys. This includes creating, editing, and deleting surveys from the system. These features are required to be accessible only by admins using some form of authentication. Requirements for editing a survey include being able to add/remove questions, reorder questions, and change the content of the questions including their type, text, and choices.
Reporting Data

The system is required to provide a minimum level of reporting through charts and or graphs to show at least a summarized view of the answers to the survey questions, through a page accessible to admins only. The customer did not have many detailed requirements for this feature and a lot of liberty was given to the team.

Exporting Data

The system is required to provide a way to export all of the saved data to a file that is Excel compatible. The system shall allow the user to select which survey they wish to export and export only that survey. This is also only accessible by admins.

Constraints

Design

1. When designing the survey we had to make sure we created something that was easily usable by children.

2. The survey will be running on touchscreen tablets. This led the team to design the main survey interface with touchscreens in mind. Touch capabilities were something both the team and the Discovery Center were interested in making use of to make the surveys more intuitive to use and interactive.

3. When designing admin features we wanted to keep them as easy to use as possible since the Discovery Center staff does not have a lot of technical experience.

Delivery of the System
1. The application’s web server is required to run on a mid range notebook computer provided by Lockheed Martin.

2. The hardware constraint also means that the application must be able to be installed on the provided computer, which runs a Windows 8.1 basic version.

Development Process

The team chose the Evolutionary Delivery process methodology to follow. The diagram above illustrates the basic flow of the process. The team focused on the first 3 steps during the first semester and did all of the iterations during the second semester. The team chose this because it’s a good compromise between waterfall and pure agile. The team felt that given a hard deadline and fixed resources, a pure agile methodology, such as scrum wouldn’t be a great fit. Evolutionary delivery allows the team to focus on the foundational items first, waterfall style. Allowing us to thoroughly capture requirements and decide on architectural stuff. The team also produced documents for each of these steps that were
shared with the stakeholders. Then the second half was done much more agile, using what we call 'cycles', for actually coding. Allowing us to develop the project in iterations.

A slight modification that we made was that we did not actually deliver the product to the customer. Given our situation of hosting, it would not have been easy to do. Instead we set up a publicly available version of the project, hosted by somee.com. We then gave demos when we finished a cycle and allowed the customers to access the somee website in their free time, in order to give us feedback. The team felt this process worked very well overall. The team also liked that we were not bound to timeboxed sprints. Cycle lengths were estimated, but remained flexible as sometimes the team went a little over or under, or had to move some features to a different cycle. This flexibility was nice to have and also worked well with the customers as sometimes scheduling demos was not easy to coordinate, due to the fact 3 different parties were involved. Another addition to the process was that the team held weekly status updates with the customers, to let them know what was going on.

The team didn’t follow strong roles, which goes along with the flexibility of our process. The process itself doesn’t specify any roles (i.e. there is not scrummaster type of role), and the only real strong role was that of the customer contact, filled by Jon Shippling. He kept up on communicating with the customers and was always on top of emails and sharing documents. Besides that, Jon and the rest of the team filled into roles that needed filling. Everyone kept everyone else organized and pitched in where needed.

This process methodology was not mandated by the customer, instead of the team discovered it while coming up with the original project plan. The customer respected our opinion and choice on the matter and in the end it worked well for everyone.
## Project Schedule: Planned and Actual

<table>
<thead>
<tr>
<th>Week(s)*</th>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Setup</td>
<td>Setup the development environments on everyone’s machine. Including having the models implemented and the database seeded</td>
</tr>
<tr>
<td>1-2</td>
<td>Cycle 1</td>
<td>Get basic flow of surveys working. Get Adult style questions working.</td>
</tr>
<tr>
<td>2-4</td>
<td>Cycle 2</td>
<td>Get basics of survey configuration done (CRUD of surveys/questions) Add exhibit additional functionality.</td>
</tr>
<tr>
<td>4-7</td>
<td>Cycle 3</td>
<td>Data reporting</td>
</tr>
<tr>
<td>7-10</td>
<td>Cycle 4</td>
<td>Create Child version of questions for children survey.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create functionality to export Survey Data to a Excel compatible spreadsheet.</td>
</tr>
<tr>
<td>10-12</td>
<td>Cycle 5</td>
<td>Finish testing, installation, and deployment</td>
</tr>
</tbody>
</table>

The team created a schedule by separating features into cycles. A cycle being based on the process methodology of iteratively developing a version that we could show our customers. A cycle is not a defined amount of time, but we tried to estimate a number of weeks we thought it would take based on what we were trying to accomplish during the cycle.
We based the estimation on what could be worked on simultaneously and considering dependencies. Also we knew some simple “grunt work” of setting up environments and such would take less time than say, developing core features.

We included some of the setup as its own week, called Setup instead of being assigned a cycle number. This is analogous to a sprint 0 in Scrum.

While our schedule was pretty solid, it definitely varied a lot. One of the reasons for this was because we didn’t account for incorporating customer feedback in our schedule. Also sometimes we started earlier than scheduled because we had time, but had to wait to call a cycle finished until we had our customer demo. Also how we deployed changed part way through, as we didn’t find until a few weeks into our second semester that we were going to be given a laptop to deploy to.

Lastly there were issues scheduling things with our sponsor and customers. Sometimes we pushed demos back a week or two due to these scheduling conflicts. This contributed to a blurring between cycles beginning and ending.

System Design

Before starting on project design, it was important to know which devices the team would be deploying to. If android tablets were used, then Android apps were a possibility. However, it was decided that a web application would be made so that there was a single unified interface for any client device. Since It wasn’t known until January which devices we were deploying to, this made this solution the best approach.

Once it was decided which platform we would go with, we had to decide which technologies we would use. Some of the team had experience with ASP.net MVC as well as CakePHP; however, Asp.net was preferred over CakePHP. The MEAN stack was another
possibility, but there was risk that the team did not want to face. It was heard that angular has a high learning curve which would put more pressure on the project and not as many requirements would be met. After exploring these different technology stacks, ASP.net MVC was chosen.

ASP.net MVC has a predefined architecture which simplified the design, made it easier to start coding, and also kept separation of concerns. Any new functionality could essentially be made into another Controller. The hardest part about design for the team, however, was designing the database and the views. During requirements elicitation, mockup UIs were created to help design the pages as well as gather requirements. Once the requirements were gathered, a proper database design was created by the team.
The above diagram encapsulates the high level architecture of the system. Though MVC kept separation of concerns by grouping similar concerns in the controller, it also allowed multiple controller actions to use the same view. This made it easy to not duplicate code, which also improves maintainability. For example, our create survey and edit survey actions used the same view to render, but simply passed in different data retrieved from the model. Models were also kept separate and somewhat isolated from the view. In some cases, other variables other than those in the model needed to be passed to the view. In these cases, view models were created to hold all the necessary data for the view and kept the backend models untied to views. Models classes are registered to a database context. Once registered, the database context then can make queries to the underlying database. The entity framework is included in ASP.net MVC which is an object relational mapper and helps make database interactions easier while also providing middleware.

Process and Product Metrics

Metrics were not given too much importance by the team. The biggest metrics we used were related to our process when it came to scheduling the next cycle. While we did have a schedule (see above) laid out ahead of time, there was more fine grain scheduling done before each cycle. This included laying out smaller tasks and seeing what we had to do for features. We made heavy use of Waffle.io during our second semester to aid in this. Waffle.io makes use of github issues and allows tracking of whatever you want. We used it for tasks/features as well as bugs.

Waffle.io has some built in metrics with its throughput graph. This was useful to use because, as we weren’t using Scrum, we weren’t creating burn down charts and measuring
velocity. This throughput graph allowed us to see exactly how many tasks we were able to do a week. This aided us in scheduling cycles as we went along as we had something to compare to. Also since github issues were used for tracking bugs, we were able to see metrics such as number of bugs and time taken to fix bugs.

Product State at Time of Delivery

The product delivered to the customer met and exceeded expectations and commitments. The team accomplished everything that was stated in the requirements and then some. This includes taking surveys, editing them, targeted design through themes, exporting and reporting data. There were several features added that were not originally discussed, such as the ability to edit themes and spinner question types.

Though not ever originally put into the requirements or committed to by the team, there are several improvements that could be made to further the product. One of the largest ones is the ability to have questions depend on or relate to other questions in both taking surveys and reporting on them. For example, the museum staff inquired if they could create a question asking a child’s age, and then based on that response pose another question. So for example children answering they are of a certain age may get a question that’s more wordy or in depth, or simply just targeted at their demographic better. Where as a child answering they are quite young, and can’t be expected to read well, could be given a question with mostly pictures.

Related to this but in reporting the data, the staff asked if they could see, for example, what museums girls liked vs boys. While it is possible to track a single submission and see all the answers one person provided via the exported CSV file, there’s not very easy way to do this.
The team decided that “related questions” type features would be too much to commit to, especially as they came up relatively late in the development of the project.

Other improvements are more general things that expand upon existing functionality. For example more question types or more types of charts in the reports. Also while themes are editable, they are only editable through manually changing a CSS file, which the team cannot expect the staff to do due to their technical knowledge. For this reason the team envisions the project having a simple GUI interface to change many of the theme related options, such as color, fonts, sizes, etc.

Project Reflection

There were many lessons learned throughout the project and improvements that could have been made. We did not have strongly defined roles, and while it worked out alright overall, the team perhaps could have benefited from people in more defined roles. Everything was done and covered but it was sort of on a reactive as needed basis. Having someone in charge to proactively do certain things could have helped. Also communication was sometimes an issue, especially for figuring out when we were going to have meetings outside of our designated time. Also communicating when we were cancelling meetings. We relying on email heavily but could have used more direct forms of communication.

For the actual development of the project, we didn’t start until the second semester to actually code. We probably could have done more, even if it wasn’t full out functionality, in the beginning semester. Really anything would have given us a headstart and been helpful. Also while the first semester was document heavy, the second semester had very little documentation. We had weekly meetings with the customer and frequent demos, but it
probably would have been helpful to have more good old fashion documentation, besides emails, to keep everyone on the same page.

There were also things that didn’t go so well that we didn’t really have control over. The largest thing being scheduling. We were dealing with a sponsor, our team, faculty coach, and the customer (museum staff) who all needed to be present for meetings. Sometimes one party was unavailable and it caused all sorts of problems. Also we relying on Lockheed Martin’s teleconference system, but if they couldn’t come to a meeting we had to re-work how we were going to call each other. Another thing that wasn’t great was that we didn’t find out we were getting the hardware that we got until late into the project. It was a couple weeks into the second semester before we got the laptop and tablets. It would have been nice to get these things earlier or to at least know that we were getting them earlier, but it was sort of last minute and out of our hands.

While plenty of bad things and lessons learned, a lot went right as well. The team progressed through the project smoothly, we all got along, and the customer was very happy in the end. We all learned a lot throughout the process and come away with a new perspective on the real world of software development.

References

● Evolutionary Delivery picture and info from here:

  http://www.Iwassoc.com/Product%20Development.htm