12-4-2014

Kitchenham, Pfleeger, and Garvin's Five Perspectives of Quality Applied to Theater Ticket Managing Software

Cortney Alyssa Wood

Kitchenham, Pfleeger and Garvin's Five Perspectives of Quality Applied to Theater Ticket Managing Software

by

Cortney Alyssa Wood

An Honors Thesis
submitted in partial fulfillment of the requirements
for the Honors Diploma
to

The Honors College

of

The University of Alabama in Huntsville

05 December 2014

Abstract

In 1994, Barbara Kitchenham and Sheri Pfleeger wrote an article applying David Garvin's five perspectives of product quality to software quality. These perspectives are the transcendental, user, manufacturing, product, and final view. In order to assess the quality of my senior project, Ticket Magic, I will analyze the application through these five perspectives of quality. In doing so I aim to increase my understanding of software quality and of how well my senior project team achieved its goal of creating a theater ticket managing software. Additionally, I hope to illustrate that while the field of technology is changing exponentially, software quality is a concept that is subjective but unaffected by this evolution.

Honors Thesis Advisor:
Dr. Richard Coleman
Lecturer

Advisor Date

Department Chair Date

Honors College Dean Date

Cortney Wood

UAH Honors Thesis

05 December 2014

Kitchenham, Pfleeger and Garvin's Five Perspectives of Quality Applied to Theater Ticket Managing Software

The field of technology is in a state of unrelenting and exponential growth, but one of the aspects that remains constant is the assessment of quality, as shown by Kitchenham and Pfleeger's five perspectives of quality from 1994. These perspectives can still be effectively and productively applied to software development today, as I shall demonstrate with my Computer Science senior project, Ticket Magic. By analyzing this project through these five quality perspectives I will illustrate their ongoing relevance as well as develop a better understanding of my own group's achievements and shortcomings in creating this software.

Before delving into the analysis of my project, it is important to understand the theory of quality I am exploring. In 1994, Barbara Kitchenham and Shari Lawrence Pfleeger wrote an article titled "Software Quality: The Elusive Target" in which they further discussed ideas presented by David Garvin in his 1984 article "What does 'Product Quality' Really Mean?". In Garvin's original article he discusses the concept of product quality, rather than software quality. He says that the lack of product quality is due to the division of quality assessment into four fields with distinct viewpoints. These fields are philosophy, focused on definitions; economics, focused on profits; marketing, focused on customer satisfaction; and operations management, focused on the manufacturing process (Garvin 1). He claims that each of these views is relevant and productive

when analyzing a product and thus attempts to combine them into a coherent theory of quality assessment.

The theory Garvin creates translates these four distinct viewpoints into five perspectives of quality, and it is these perspectives that are elaborated upon by Kitchenham and Pfleeger in the article which is the basis for my quality analysis. These are the transcendental, user, manufacturing, product and final perspectives. First is the transcendental perspective, which is "something toward which we strive as an ideal, but may never implement completely" (Kitchenham 13). In other words, it is the most ideal implementation of our software. The second perspective is the user perspective, an assessment of "product characteristics that meet the user's needs" which "evaluates the product in a task context" (Kitchenham 13). Thus, it is an analysis based on the use cases of the product. Third is the manufacturing perspective, which "examines whether or not the product was constructed 'right the first time'" typically trying to reduce extra work during development and after product delivery (Kitchenham 14). The product perspective looks at the engineering aspect of the product by "considering the product's inherent characteristics" and

Summary of Requirements for Ticket Magic
System will be portable and compatible with Windows 7/8 and Mac OSX.
System will include facility to get new users acquainted with the application.
System will be default-driven for purchasing a pair of tickets for a single show.
Customers can purchase tickets for a single event or a season, online or at the door, and may pay for these tickets with cash, credit or check.
Tellers may refund tickets or exchange them for a ticket of equal or differing value.
Season ticket holders' information will be retained, modifiable, and exportable.
Organization managers for each theater group may create events as well as provide custom prices for the theater group as a whole.
Seats are identified by section, row and seat number.

by “measuring internal properties” (Kitchenham 14). This means judging how well an application is designed. Lastly is the final, or value-based, perspective. This perspective analyzes quality based on what viewpoints or aspects are most important for the specific application at hand. These vary based on the context for which the application is developed and are useful as a way to provide a general analysis. With these five perspectives of quality, I aim to better understand how well my team and I have implemented Ticket Magic and learn more about increasing software quality.

In addition to the theory behind my analysis, in order to properly assess the quality of our software I must first discuss what requirements the software is expected to fulfill. Our product, Ticket Magic, is an application for facilitating the purchasing of tickets for shows at the Civic Center Playhouse and Civic Center Concert Hall. Therefore, it includes a user interface and functionalities for customers purchasing tickets, theater groups performing events, tellers working the door of the events, and the administrator running the venues. The following table lists a summary of the requirements for Ticket Magic:

With these requirements in mind, it is easier to understand and fulfill the primary purposes of this software for the customer: to make it easier for volunteers to manage the ticketing process and to use a common software package for all organizations working with the Civic Center. Both of these purposes result in reduced time spent teaching new volunteers how to use the ticketing software. In analyzing the software quality of Ticket Magic, I will be comparing its functionalities with these requirements and purposes.

Table 1 - Summary of Requirements for Ticket Magic

The first perspective of Garvin's theory is the transcendental: the ideal implementation of our software. In order to apply this perspective to Ticket Magic, I will discuss which features would be in the ideal application and whether or not missing these features negatively affects our fulfillment of the requirements. This discussion should reveal the quality of my software by comparing it to the best possible implementation and analyzing the discrepancies. Because our project has been completed within a semester, this section of analysis will be the most in depth. There are many design choices that were heavily affected by both our time constraints and additional time spent learning Ruby on Rails, our language of choice which none of us knew prior to this project. For this reason discussing the shortcomings of our implementation is an effective way to judge which features we sacrificed in order to finish our project on time and whether these choices allowed for the best fulfillment of our requirements.

As a software application that will be used by many different groups, one of the main aspects our implementation could improve upon is customization. The ability of an organization to have a logo and cast list as well as posters for individual events would be a great addition to Ticket Magic. However, lacking these features does not affect the requirements for an organization: we still have custom price groups and organizations can create events. It might be better if the organization could have custom prices not just for the entire group but for each individual event, with these prices being selected upon event creation. Additionally, while all the price values are customizable, there is only one field for a custom price group with a custom name and price. While our current implementation of price groups is adequate, I believe prices for each event and multiple custom price groups would be a better fulfillment of the requirement.

The basics of event creation are implemented - there is hosting organization, event name, description, time, and date. However there are several aspects of this process that could be improved. Aside from the aforementioned ability to add posters and custom prices, it would be more user-friendly if there were a visual calendar for selecting the start and end time for an event. The current implementation using drop-down menus is not necessarily difficult to use, but seeing the dates laid out as a calendar would be easier and more practical. Additionally, the ability to create a show and select multiple performances would have been more user-friendly. With the current implementation the user must create a new event for each performance. Considering the context of theater revolving around multiple shows, this is a poor implementation and negatively affects our fulfillment of the requirements relating to usability.

Additional improvements could be made in the ticket purchasing procedure. Both customers and tellers must work through a multi-page ticket selection process which involves drop-down menus accompanied by seating charts for reference. These menus only display available seats for each option - area, section, row and number. It would be much easier if this could be reduced to a single page which uses an image map of the seating chart, allowing the user to visibly see which seats are sold and which seats are available. Then the user would be able to click on a seat to purchase it. While our implementation fulfills the requirements by allowing a user to purchase seats and defaulting to a pair of seats, it is not very user-friendly. If there are many seats purchased already, finding two neighboring seats could be very difficult, requiring the user to go back and select different sections and rows. I believe our implementation of ticket purchasing is poor and this can be attributed to the lack of time to learn about and implement the image mapping. As a group, we underestimated the amount of time a seating map would take. This error

was a learning experience as software developers which led to a better understanding of the design and development process.

The last area that is largely different from our ideal implementation is season tickets. Only an organization manager may purchase a season ticket and the process is borderline painful. A customer must call the organization manager and, referencing a seating chart, verbally request a season ticket by asking for a specific seat. A seat must be requested for both the Playhouse and the Concert Hall - the two venues in the Civic Center. This is so that the season ticket holder will have a seat for events in either venue. A better version of this implementation would be to at least check if the organization has events in each venue and only request a seat for venues where this is true. Once the user has requested the seat, the organization manager must manually type in each part of the identification string separately: area, section, row, and number. These must be entered for both venues, and then the manager submits the form. If either one of the seats is already purchased for any of the currently existing events then the manager must input a different seat. This process is repeated until a pair of available seats is entered. The problem with this implementation is that it could be very tedious and is hardly user-friendly. A much better method would be to duplicate the seating map used for ticket purchasing, but any seat that has been purchased for any show would be displayed as taken for season tickets. Additionally, the actual design of season tickets is not a unified object, but rather a collection of tickets. When a season ticket is purchased, what actually happens is that the system goes through each event with a distinct name and purchases a single ticket in the chosen seat. The user is then marked as a season ticket holder. Ideally, a season ticket would be implemented as its own data structure which con-

tains references to the individual tickets. Like the issue with the ticket purchasing implementation, this shortcoming is a result of a lack of time combined with poor work estimations.

In making design choices we were forced to leave one requirement unfulfilled. Due to the complexity of a practical implementation, we were unable to provide an import feature as requested by the customer. With more time to work on this project this feature would definitely be feasible. However, due to prior choices made for a speedy completion rather than a clean implementation, the process of importing data became infinitely more difficult. The primary data to be imported and exported is season ticket data. Because season tickets are implemented as disconnected individual tickets, importing a list of season tickets becomes a complex logic problem which our team simply did not have time to solve.

Ticket Magic Use Cases				
Use Case	**Admin**	**OM**	**Teller**	**Customer**
Create a customer account.	-	-	-	✓
Create an organization manager or teller account.	✓	-	-	-
Create an event.	-	✓	-	-
Modify and create custom prices.	-	✓	-	-
Purchase tickets.	-	-	✓	✓
Purchase season tickets.	-	✓	-	-
Exchange or refund tickets.	-	-	✓	
Export season ticket holder information.	-	✓	-	-
Import season ticket holder information.	-	X	-	-

Overall, the shortcomings of Ticket Magic when compared to the ideal implementation all come down to a lack of time. It is very likely that if the team did not have to learn Ruby on Rails before designing and implementing this software we would have been able to provide a much more clean and practical application. With this obstacle in mind, I believe we did an adequate job fulfilling as many requirements as possible in the allotted time frame.

The user perspective is the second viewpoint from which Ticket Magic will be analyzed, an analysis which means stepping through the multiple use cases. This application will be used not only by the customers who have requested it but also by every one who wishes to purchase a ticket for the Civic Center online or volunteer to work at the door of events. Thus the perspective of the user and how well each use case is handled becomes a critical aspect of quality.

There are four types of users which will be interacting with the Ticket Magic application: administrator (admin), organization manager (OM), teller and customer. These user types have overlapping use cases, thus this assessment will be divided by individual tasks rather than user groups. The most efficient way to analyze our software from this perspective is to show each use case in a table with a column for each user group. If the group requires the ability to perform this task, then there will be a checkmark in that user group's column if the software supports the task. If the software does not support the task there will be an X. If the user group does not require this task, there will be a hyphen. This task breakdown is shown in the following table:

As discussed during the transcendental perspective analysis and shown in the table above, the only required task which is not implemented is the ability to import season ticket holder information. All other use cases are addressed and functional, meaning the majority of the users of the

Table 2 - Ticket Magic Use Cases

system will be able to perform all necessary tasks. This reflects positively on Ticket Magic's fulfillment of the requirements from the user perspective.

The third viewpoint for assessing quality is the manufacturing perspective. This view is primarily concerned with the logistics of software development by analyzing whether or not the software was constructed "right the first time" (Kitchenham 14). In the context of our project this view is the least relevant, since Ticket Magic is a semester-long project with no need for maintenance or reworking after the final delivery. Therefore this section of analysis will be the most brief.

There are design choices discussed in previous sections which negatively affect the quality of our software from the manufacturing perspective. Because the team was focused on finishing as much of the project as possible in a short amount of time, there are features which could have been implemented more efficiently. If this project were to move into a second phase of development, the entire second phase would likely be spent fixing the design choices from this semester which favor speed of implementation over quality of design. For this reason, Ticket Magic has very poor software quality from the manufacturing perspective, though this viewpoint is arguably irrelevant in this context.

The fourth perspective, product, will analyze how well we utilized the knowledge gained from our semesters as Computer Science students as well as how well we took advantage of the powers of the Ruby on Rails programming language. In our initial design of the application, we utilized object-oriented and database design skills. Unfortunately, the final implementation was very different from his design and lacks the quality we originally planned. However we did use

many of the strengths of Ruby on Rails to improve our experience as developers and the experience of the users of Ticket Magic.

Our implementation using Ruby on Rails consists of two primary parts: controllers and layout files. The controllers contain all of the functions which manipulate and create objects such as tickets, events, and seats. The layout files contain the HTML code for the website and the Ruby code which links the website to the controllers. Thankfully, Ruby has built-in functionality for testing these links between the controllers and the website as well as the functions within the controllers. These tests can detect if the functions are not working, though not necessarily if they are working incorrectly. We still needed to manually test many of the functionalities of our application, but the built-in testing within Ruby on Rails greatly reduced our integration testing.

Another great advantage of using Ruby is access to many open-source gems. Gems are the foundation of Ruby, similar to libraries in C++. We were able to reduce our workload by finding gems for certain functionalities, meaning we did not have to write custom implementations ourselves. For example, we used a gem for password encryption rather than attempting to write or implement an encryption algorithm on our own. We were then able to devote our time to the core functionalities of the program rather than focusing on learning about encryption, a topic which none of us were very familiar with. Additionally, we used a gem for form creation. Since our application involves users creating and modify many objects in the database, the ability to easily create and implement forms greatly reduced our workload.

Ruby on Rails also has built-in database integration, further reducing the trial and error of traditional databases. Using terminal commands we were able to create all of the database tables for our application, automatically creating and linking these tables to their corresponding con-

trollers and layout files. This ability combined with the previously mentioned built-in testing almost eliminated integration issues and testing.

As previously mentioned, we were unable to implement our original designs for the database. In our team, only one member had previously taken a course on database systems, while two of the members were enrolled in a database course during development. Therefore, while we were capable of designing a robust and viable database, actually implementing this database became difficult. We encountered many obstacles concerning communications between database tables. While we understood which relationships needed to exist, due to the complex nature of these relationships and our inexperience we had to reduce the design of our database to a very simple implementation. Almost every table communicates through the user table, since tickets, season tickets, and events all need to interact and are all owned by a user. Taking into consideration our inexperience with databases and the time constraints for the project, I believe we made good design choices. The database is functioning and relatively secure, and through creating the preliminary design we solidified what knowledge we have of relational databases, even if we were unable to implement that design.

The last perspective to discuss is the value-based assessment. For this analysis I will determine which aspects of quality are most important and relevant in the context of our application then discuss how well we performed in each of these areas.

The most valuable areas of quality for Ticket Magic are usability and efficiency. Our product is built for both trained users (the administrator, organization managers and tellers) and completely new guest users (the customers). Thus we must have a system that caters to both of these groups. In order to do that, our application must be very user-friendly. The application

website has a help page detailing every use case and which steps to take in order to complete the related task. This help page exists to familiarize new users with the application, as requested in the requirements for the project. Its existence greatly increases the usability of our application by providing step-by-step instructions for each task, reducing user frustration or misunderstanding.

There are a few factors negatively affecting usability, primarily the process by which a user purchases tickets, both regular or season. As discussed during the transcendental analysis, this is a multi-page process that results in a series of trial and error from the user. Additionally, there were several occasions where, as a team, we made choices assuming the user would understand how to use the system. These choices were primarily made in regards to administrator or organization manager tasks and were only made due to the lack of time to implement more automatic or foolproof processes. An example would be organization creation. When the administrator creates an organization we do not check if an organization with the same name already exists. Writing the functionality to verify every new organization name is distinct would have taken too long and might have resulted in not satisfying other requirements. In this particular case, the result is that there may be two organizations with the same name. Most mistakes by the user will not cause the system to crash, it is just more easy than it should be for a user to make these mistakes. For these reasons, our application has room to improve in regards to usability, but overall is still a decently user-friendly system considering the time we spent building it.

The second aspect of quality that is extremely relevant in the context of Ticket Magic is the efficiency of the application. Ticket Magic will be used by everyday customers, not just trained employees. Therefore it is unreasonable to expect that every customer will be patient if the a process is slow. Thankfully, every process within Ticket Magic is essentially a database

query created by the Ruby interface. For a customer, these queries are showing events and creating tickets, both of which are simple and fast. The only queries which are notably slow are those which create events, primarily events in the Civic Center Concert Hall. This venue has almost 2,000 seats which need to be created, a process which takes almost a full minute. The only users who will experience this query are organization managers. While ideally no user will need to worry about inefficient processes, if the only user who does deal with slower loading times is an organization manager then the average user should still be satisfied. We have also implemented loading animations on every page to prevent the user from mistakingly thinking the system is frozen or crashing. The combination of this loading animation with the typically speedy database queries results in an acceptable level of efficiency for most users of the system.

Combining the analysis from these five perspectives of quality, I have thoroughly assessed Ticket Magic's software quality according to my own standards. Some aspects of this analysis are skewed due to the context of the application. Rather than analyzing a professional and ongoing software development process I have analyzed a condensed and finite process. For this reason the discussion of the ideal perspective becomes critical, highlighting the shortcomings of our implementation under the time and resource constraints of a semester project. Thus our choices as a design team can be critically assessed in regards to their affects on software quality and requirement fulfillment. The second most revealing perspective is the user perspective, illustrating how adequately the needs of our users have been addressed. The manufacturing perspective is mostly irrelevant to our finite, single-iteration development process. The product perspective highlights how well we adapted to using a new language, Ruby on Rails. Our ability to utilize many of the core features of the language not only speaks to our ability as developers

but is also apparent in the final product. Lastly, the value-based perspective elaborates upon the software quality in the context of the project. Because of the vast amount of untrained users, both usability and efficiency are extremely important aspects of quality and for the most part are appropriately addressed within Ticket Magic. Overall, keeping in mind the context in which this system has been developed, I believe my team did an adequate job producing a decent quality product for our senior design project.

Works Cited

Garvin, David. “What Does ‘Product Quality’ Really Mean?” *Sloan Management Review* 15 Oct. 1984: 25-45. Print.

Kitchenham, Barbara and Shari Pfleeger. “Software Quality: The Elusive Target.” *IEEE Software* Jan. 1996: 12-21. Print.