Elsa M. Rodriguez Plaza

Student ID: 903453539

Professor Mark Moss

Software Architecture and Design (CS6310)

September 30th, 2018

Assignment 4: SWEBOK Scavenger Hunt (Pick Three Snippets & Elaborate)

The relevance of documentation, changing requirements and a system's users

As software architects we tend to focus on the technical side of a system's design, we get excited about using new technologies, new design patterns, the different steps we need to follow to get the product finished, among many other things. However, we often neglect one crucial responsibility, which is to think about what the user expects from the software. This is the main reason why I agree with the statement: "Thus, all software design and development must take into consideration human-user factors such as how people use software, how people view software, and what humans expect from software." (Bourque and Fairley 13-22)

Engineers have to think how to automate people's manual processes, but if we do not take into consideration the personal side of things we not only develop an unusable system, but we also waste time and resources.

During my professional career, I have faced many challenges, but I recall specifically this one time, when my team was reimplementing a legacy system. We developed the system's backend as a micro services architecture and the product's frontend as single-page application. Even though, we had a robust software, developed

using cutting-edge technologies and implemented all the features present in the legacy system, we got negative feedback from the clients. The reason why this happened was because we never inquired on the fact that the clients wanted to have two simultaneous views of a data dashboard. By not clarifying this requirement on the users' visual preferences not only did we miss a crucial use case, but also had to delay the release by four weeks.

Another important factor to consider when designing software is how to handle the error messages shown to the human-users. As an example; a few years ago, my team and I implemented a system for international users and we created an Internationalization service. This service was reused by other systems for translating error messages displayed to the client. The logic of this service was to detect the users' location and based on that translate the message received from other services to the language spoken in the region that the user was at. The Internationalization service had a fundamental design flaw: users can travel to other countries where their native language is not spoken. Therefore, we ended up with situations in which we showed German error messages to British users. The usability of our system was compromised by the fact that we never considered a common situation in people's lives. People travel to other countries and still wish to see the user interface in their preferred language.

When designing software it is very important to consider all possible cases, circumstances, and most importantly the human-user necessities for whom the software was built; otherwise, we might miss important features to be added during the design and implementation phases.

Whereas, understanding the users' requirements and interactions with the systems is a vital step during the Software development process, documenting these requirements, the design decisions, the technologies that will be used to develop the system, among others is also crucial to successfully deliver a great product. So, while reading the Guide to the Software Engineering Body of Knowledge (SWEBOK), I came across an statement that I consider summarizes my opinion on documentation: "One of the most important activities in design is documentation of the design solution as well as of the tradeoffs for the choices made in the design of the solution." (Bourque and Fairley 15-10).

As per my experiences, not documenting the design introduces multiple problems when developing a solution. Firstly, if the engineers in charge of the design are moved to another project, whoever continues working on the project will not have a point of reference if documentation is unavailable. This situation could lead to having to redesign the whole system.

Secondly, there are quite a lot of different technologies for solving the same problem. However, not all of those technologies will be suitable for every use case. Executing proof-of-concepts and documenting the results helps architects to evaluate which technologies are better equipped for solving certain problems.

Finally, transforming the requirements into a detailed technical language that engineers fully understand is also extremely important. This could be an iterative process, where the architects first look at the requirements and as they continue to understand the system better, they can elaborate on the problem statement and understand how the system should behave as a whole. Understanding how the system behaves as whole and

documenting this behavior helps everyone who works on the project to have a broader picture of the software and it also allows to better estimate the effort.

I recently worked on a project where the only documentation available was the source code. I remember that the program used a Redshift database provisioned by Amazon Web Services and the queries ran so slowly that the users had to wait for over five minutes to see the report data. Since there was no documentation, I spent several hours trying to understand why this type of database was agreed upon when there were other alternatives that were not only more efficient, but also designed specifically for the type of data used handled by the software. The conclusion of my research led me to reimplement the database schema using a time series database. The reason why this happened is because the previous designers did not evaluate the best technologies for the type of data as per the system's use cases, they did not document the design phase; instead they jumped directly into the code without fully understanding the problem.

In the professional industry, engineers have other functions besides developing code. They have to present to stakeholders the status of the projects they are contributing to, as well as the decisions made throughout the design phase. Often these meetings are scheduled at the last minute, so if there is not documentation available and the software is not ready to be demonstrated, there will be no visual aid, nor visible results to present. This is another reason why documenting our products is crucial during the Software development process.

During the lifespan of a project, the requirements will always change, as stated in "Perhaps the most crucial point in understanding software requirements is that a

significant proportion of the requirements will change." (Bourque and Fairley 1-13). There are many reason why this situation might occur. For instance, the customers could ask for new features that impact the definition of original requirements. Another reason that could potentially impact the redefinition of requirements would be changes in regulatories policies enforced in the country where the software will be distributed.

Recently, I came across a situation where our clients were spending a fortune on cloud computing. Thus, we were asked to implemented a solution based on our customers' cloud hourly usage that would recommend the optimal number of hours to buy based on historical data.

Even though, we processed historical data to obtain the purchased hours as per the original requirement, there was a change requested by the customers to improve the accuracy of the calculation. The clients wanted the software to process the owned hours not by using historical data, rather by using the cloud provider's application programming interface. Situations like this are not uncommon in the professional world, engineers need to know that the requirements will change over time to adjust better to the clients' needs.

I came across another example of requirement change when the General Data Protection Regulation replaced the Data Protection Directive in Europe. The General Data Protection Regulation changed the way in which the European resident's data is handled by enforcing regulations on privacy and rights. (GDRP). At the time, we were implementing a Big Data Pipeline that processed personal data from European residents. The original requirement was that the data was going to be hosted and subsequently

processed in our Oregon Data Center in the United States. The data was going to be transferred over using an Extract Transform Load implementation. As a result of this regulation, the requirement on data transference changed, so the Extract Transform Load implementation had to be redesigned to meet the regulation's requirements on data security and privacy.

In general, Software Engineering is a discipline, in which requirements are always evolving, so they can better accommodate users' needs. The main reason why requirements exist is so that they can help architects define how the system should behave and the interaction between the clients and the software. Fortunately, documentation is a great resource for expressing our thoughts on a system's design, as well as a great aid when it comes to showing the progress to our stakeholders.

Works Cited

Bourque, Pierre, and R. E. Fairley. *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, 2014. 30 Sep. 2018

"The EU General Data Protection Regulation (GDPR) Is the Most Important Change in Data Privacy Regulation in 20 Years." *Key Changes with the General Data Protection Regulation – EUGDPR*, eugdpr.org/. 30 Sep. 2018