Prospectus

Automated Course Advising Assistant

(Technical Report)

From "Design For" to "Design With": Community-Designer Relationship and The Appropriate Design Paradigm in Creating a Digital Story-telling App

(STS Research Paper)

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Sean Gatewood

Fall, 2019

Department of Computer Science

Signed: Sean T. Gatewood

Approved: Tsai-hsuan Ku Date 12092019

Sean Ferguson, Department of Engineering and Society

Approved: Date 4 Dec 2019

Luther Tychonievich, Department of Computer Science

Introduction

One of the most important and difficult challenges in software engineering is the process of defining and maintaining the software's requirements. The virtual world allows for endless innovation often limited only by the engineer's imagination and time. However, this unstructured power also creates infinite opportunities to produce exciting, functional software that doesn't solve the original problem that it was meant to solve. Software requirements attempt to provide a clear definition of what the software needs to accomplish, and defining these requirements requires a clear understanding of both the stakeholders and the problem they are actually trying to solve.

Furthermore, the challenge becomes even more difficult when designing software for large communities. Here, without an individual customer to consult, many of the traditional requirements elicitation techniques start to break down, usurped by techniques of community engagement. Designing for a community is much different than designing for individuals, and requires methods and efforts to understand both the community as a whole and the spectrum of diversity within.

This topic is particularly interesting from an STS standpoint, since it embodies a very real intersection of technology and society, while also serving as an application of community engagement within the technical world. The topic was born from my team's blueprint for an application to allow community members in diverse cities like Charlottesville to share their stories and combat the idea of invisibility within cities. Engineering this application would really be engineering a small social movement, so its success would depend on whether or not the platform actually met the community's needs.

The topic is also interesting from the standpoint of my capstone research group, as we are designing an automated course advising assistant for the UVA student community. In order to accommodate as many students as possible, we must first have a deep understanding of the diversity of needs within the student community.

Technical Report

Here at the University of Virginia, in the weeks leading up to course registration, all students must meet with their academic advisors to help decide what courses they should take in the following semester. The optimal decision for a student depends on many factors, such as the student's career interests, course history, personal preferences, extracurricular commitments, enrollment priority, and learning style. Although there are often several schedules that could be considered optimal for a given student, we feel that much of this decision is deterministic, and could be automated if all the data is brought together. We hope this will help standardize advising at UVA, and remove much of the stress that comes from weighing all these factors within the human brain.

Designing this automated advising tool has proven to be an interesting challenge, especially due to the Family Educational Rights and Privacy Act (FERPA). This federal law, which applies to all institutions that receive funding from the Department of Education, protects the privacy of students' academic records. Although our technical advisor is authorized to handle this data, we decided it would be best from a security standpoint to not store any student's grade information anywhere in our system. Additionally, we decided not to base our decision algorithm on grades, other than excluding failed courses in a student's course history. We figured that grade information, in addition to being one of the more sensitive pieces of FERPA information, is too narrow of a metric to measure a student's performance and interests for course registration purposes. We will still store student IDs, lists of completed courses, and academic plans, which could all be considered FERPA-protected information.

Additionally, the tool must be readily available for all students to use, regardless of their personal computing resources. For this reason, we decided that a web app would be the best platform for this tool, taking advantage of all the containerization benefits of browsers. In other words, using a web app would likely require the least amount of setup for our users, who may have variable amounts of technical expertise, as well as variable operating systems.

At a high level, our technology stack begins with React on the front-end. We decided to use React mainly due to the curiosity of our team members, but also due to its ability to modularize website components and its established reputation. (Using a common framework like React helps in the maintenance of software, ideally allowing anyone with a background in the framework to understand the system and make changes if necessary.) On the back-end, we have a Django application running in a Docker container. The Django application (a large Python program) handles all of the requests to our website, delivering the React interface to the user, as well as handling any operations that need to GET or SET data in the database. The Docker container defines an environment in which our application runs, effectively automating the task of setting up any systems we run our final product on. (A computer only needs to have Docker installed to build and run our entire system.) In summary, when a user accesses our website, our Django application responds with the HTML, Javascript, and CSS that the user's browser renders into a React application. When using this application, if the user performs an action the requires data to be retrieved from or set within the database, the React application completes this task by making additional HTTP requests behind-the-scenes to our Django application, which handles the requests.

STS Thesis

Introduction. One of the greatest challenges in software engineering, arguably more difficult than the actual software development itself, is determining *what* exactly to develop. Answering this question requires a deep understanding of the user and their problem(s) that you expect to solve or help alleviate. However, this task becomes even more difficult when designing for a large, diverse community of users. Understanding just one kind of user is difficult enough; How do you begin to learn about the needs of a large number of people, while also taking into account the complex interactions between them as a community? What methods or design paradigms can we use to form this deep understanding?

Literature Review. Perhaps we could still consider a classic user-centered design approach, such as those used in typical industry settings. Defining clear and correct requirements is so vital to software engineering that a whole field of research has been dedicated to the process of "Requirements Engineering." Nuseibeh and Easterbrook describe some of the challenges in this realm. The requirements should not only define what the software should do, but also why the proposed feature(s) would be useful or necessary. Along these lines, it is often important to verify that proposed requirements are actually what the customer wants, independent of their actual claims. Finding the true requirements of the system requires a deep understanding of the stakeholders involved, which even the stakeholders themselves may not have. It is a social challenge that ties in pillars of anthropology, sociology, psychology, and communication. It is a prominent example of an interface between technology and society (Nuseibeh & Easterbrook, 2000).

Software Engineering textbooks often offer methods for "requirements elicitation," the process of communicating with stakeholders to generate requirements. These often define interactions between the engineers and the customer. For instance, one method is to interview potential customers and work on defining requirements directly with them. Even with the customer in the room, this process still carries the aforementioned challenges, and textbooks often provide tips for carrying out this process, such as iterative prototyping (Leach, 2016). Some textbooks even suggest highly-structured rule-based approaches, such as Joint Application Design (JAD) and Quality Function Deployment (QFD), which are intended to aid the sociotechnical interactions by providing defined structures for meetings and discussions. Some textbooks recommend approaches beyond interviews, such as having an engineer shadow the stakeholders in their current environment to understand their needs at a more personal level (Laplante, 2007).

However, these design methods may begin to break down when we remove the idea of a typical "customer." Most of the aforementioned techniques were developed for software engineers to design a system that was requested by a specific customer, as is often the case in the software industry. But what about novel software that nobody specifically asked for? What if the target user group is very large? How does one design software for an entire community?

Some industry methods do exist to elicit requirements from a large user group, such as developing "user personas", which are meant to break up the community into a few particular user groups, represented like archetypes. However, it is impossible to accurately represent a continuous spectrum of diversity with only a few discrete boxes (Khan, Liu, Wen, & Ali, 2019). For instance, in developing a platform for

community members to share personal stories and histories within a diverse city like Charlottesville, engineers would need to consider that the crowdsourced content could come from a wide array of people, all with different backgrounds, experiences, and potentially different political views. Some of the stories could be fake, false, or even offensive. This fact may bring to mind features of moderation, even though these features may actually break the requirements of some individual users. Here exists a common theme: Designing for a community is much different than designing for individuals.

Designing for a community also requires accommodating a wider diversity of users. For instance, in designing an automated course advising assistant for an entire student community, we must accommodate a wide spectrum of learning styles, spatial reasoning skills, short-term memory capacities, etc. We must also accommodate students with disabilities. We have already taken steps toward this goal by adding attributes of Accessible Rich Internet Applications (ARIA) to our application, which are additional components that make the application more accessible for people with assistive technologies like screen readers. If we want our tool to have a positive impact on the student community at UVA, we must ensure we have an understanding of that community.

Additionally, we must be careful in reducing our interaction with the community to an engineer-customer relationship. As we learn in community engagement, the framing of any service relationship matters, and should be grounded in respect rather than superiority or pity. As Norman and Spencer say, we must design "with" the community, not "for" the community (Norman and Spencer, 2019). Additionally, applying a user-centered design for a community project would assume that everyone in the community would be willing to use the product, overlooking potential reasons why others may be less inclined to participate.

Community-Oriented Design. Therefore, perhaps a user-centered design mindset is not entirely appropriate. Instead, we might be able to adopt a more community-oriented design mindset. To develop a useful software product that has a positive impact on the target community, the engineer(s) must have a deep understanding of the community members' needs, experiences, abilities, sensitivities, resources, interactions, and culture.

Some methods exist to gain some of this understanding. Human-Centred Design focuses on designing from a mindset of empathy. In order to design something for a person, you have to first put yourself in their shoes. For instance, Dam and Siang argue that the failure of the Google Glass came from a lack of empathy. Although the technology was revolutionary and exciting to many people, nobody could imagine using it in public (Dam and Siang, 2018). When making important design decisions, such as the wearable concept and voice-activation, Google failed to consider how using these features might make people feel in a social setting. The social factors of using this technology were external to the technology's technical requirements and involved interactions within the user community, showing that a traditional user-centered design approach could not have anticipated this flaw.

However, designing from a mindset of empathy is not without its flaws. As Higashi points out, empathy can bias us too much towards individual users, putting individual needs before the needs of the community. If we get to know the needs and suffering of a few individuals at a personal level, we may

feel more inclined to put their needs before the needs of the rest of the community (Higashi, 2017). Additionally, community members' knowledge of the problem may not be entirely correct, and may be directed more toward the symptoms of the problem, rather than toward the root of the problem (Norman and Spencer, 2019).

Future Work and Timeline. More research needs to be done on how to best go about this process of designing for a whole community. My group is in an excellent position to do so with our particular project. In designing our application, we could try experimenting with existing practical techniques and invent new ones, to discover which overall approach; user-centered, human-centered, or a mixture of the two; is best for engineering software that reflects an understanding of the community's needs.

(The specifics of our timeline are currently being determined, as our section is undergoing a bit of restructuring.)

Bibliography

Dam, R., & Siang, T. (2018, December 1). Design Thinking: Getting Started with Empathy. Retrieved December 2, 2019, from https://www.interaction-design.org/literature/article/design-thinking-getting-started-with-empathy.

Higashi, T. (2017, August 9). Empathic Design: What is it good for? Retrieved December 2, 2019, from https://medium.com/tradecraft-traction/empathic-design-what-is-it-good-for-f665563bcc92.

Khan, J. A., Liu, L., Wen, L., & Ali, R. (2019). Crowd Intelligence in Requirements Engineering: Current Status and Future Directions. *Requirements Engineering: Foundation for Software Quality Lecture Notes in Computer Science*, 245–261. doi: 10.1007/978-3-030-15538-4_18

Laplante, P. A. (2007). *What every engineer should know about software engineering*. Boca Raton, FL: CRC.

Leach, R. J. (2016). *Introduction to software engineering* (2nd ed.). Retrieved from http://index-of.co.uk/Engineering/Introduction to Software Engineering.pdf

Norman, D., & Spencer, E. (2019, January 3). COMMUNITY-BASED, HUMAN-CENTERED DESIGN. Retrieved from https://jnd.org/community-based-human-centered-design/.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: a roadmap. *Proceedings of the Conference on The Future of Software Engineering - ICSE 00*, 35–46. doi: 10.1145/336512.336523