Software Engineering Research Report and Case Study Analysis on Automated Content Moderation

Christopher Morcom
CS 487 – Software Engineering
9 August 2019

Illinois Institute of Technology
Chicago, Illinois 60616
morcom01@gmail.com

Abstract – This paper reports on a case study that automates content moderation to an extent, describing it from a software engineering perspective that outlines specific challenges, concerns, and best practices of the system's development.

Index Terms – Software (SW) Engineering, Moderator, Semi-Automation, Automation Development

1 INTRODUCTION

The internet is a massive place containing petabytes of data that has been reviewed by man or machine, and sometimes both. As the rate of data being posted online drastically increases, we find an increasing need to effectively moderate user-generated content. This study talks about [1] from a software engineer's perspective.

This report is organized as follows: Section 2 summarizes the results of the case and whether is was quantifiably "successful" from a SW Engineering perspective. Section 3 describes the specific challenges noted by their group during each phase of development. We conclude Section 3 in an overall context and how it relates to SW Engineering before moving to Section 4. The next section talks about areas of concern and "best practices for automation development. Section 5 discusses one of the tools used for automation development. Finally, we state our overall findings and opinions in or conclusions in section 6.

2 CASE STUDY SUMMARY

With issues in the volume, velocity, and variety of data being shared over social media, the need for automating data processing emerges. The study [1] talks about how current machine learning systems process data with a "human-in-the-loop" approach which is where algorithms will occasionally consult humans for feedback and correction. The issue with this is that humans are often presented data out of context by the system, making the current systems that are data-driven redundant since context of identified information is lost.

The study mentions the "human-***is***-the-loop" approach to "semi-automate a manual content moderation workflow where human moderators take a dominant role." The study analyzes content generated about natural disasters, conflicts, and benign data for testing. Rather than the former method, where human moderators must gather context manually, the study's prototype uses machine learning to seek the context and meaning of data and "provide human analysts with suggestions regarding the relevance and categorization of collected data.

During testing, the results found that although the system accomplished in providing the user relevant information, with relatively low precision since the amount of information returned is too high, the system needs more focus on categorizing information by priority, where more relevant information is displayed first. Additionally, most organizations tend to favor frequently used sources of information, which results in an issue of building trust and need for this new system.

Since social media is mostly a subjective process, quality control is also a concern before data is sent to users for review. For instance, information about a natural disaster would need to have a different context from a conflict. However, on the positive side, the prototype successfully accomplishes its goal with "improved moderation quality and higher flexibility" while being extremely compatible with users and how their work is completed.

3 CHALLENGES THROUGHOUT DEVELOPMENT

This section will break down the difficulties noted by the authors during each stage of development.

For the research stage, the writers used the Design Science Research paradigm *(fig. 1)* which is commonly used for Information Systems (IS) in solving relevant problems, prototyping, and contributing to the knowledge of socio-technical artifacts [2].

Furthermore, the study uses IS research paradigms to move a theoretical knowledge base to practical development *(fig. 2)*. In contrast to the standard model, the writers use data-driven research to substantiate this development. Overall, the only real challenge here is data validity and quantity, which may be seen in later stages – a common issue in this stage in terms of SW engineering.

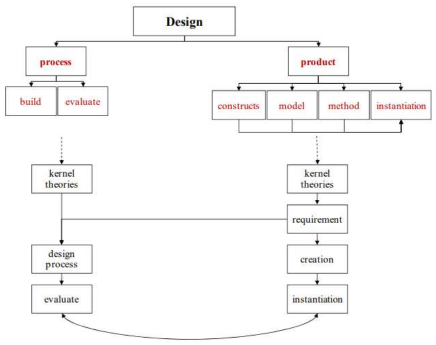

Figure 1: **DSR Framework Diagram**

During requirements specification, the writers focus heavily on workflow specification. Using previous works on automatic, hybrid, and manual workflow, the writers were unable to find a way to define requirements to fully automate their system. The writers faced a huge challenge in defining requirements that would not bottleneck the system on delivery to the human moderator, while maintaining the purity of the data. This led them to define a hybrid system, that is based on the manual workflow of the GDACSmobile system they analyzed *(fig 3)* [1]. The study uses four defined requirements for the new moderation workflow of their system. In summary, the largest issue here was defining specific requirements for the system that allow finding viable candidates for tools that help in creating the system.

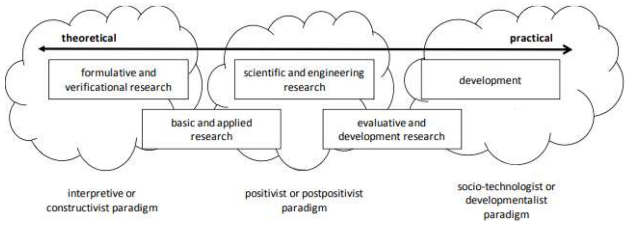
theoretical
practical
formulative and verificational research
scientific and engineering research
development
basic and applied research
evaluative and development research
interpretive or constructivist paradigm
positivist or postpositivist paradigm
socio-technologist or developmentalist paradigm

Figure 2: **Perception of Research Objectives and Methods**

Little is known about the design and development stages; however, the study reflects the challenges in each through a technical standpoint. The limitations of the software, at a fault of not enough scope defined in the requirements were that the system has no measure of trustworthiness, it returns too much data to its human users, with low precision, and the data is not sorted based on relevance/context, nor is a requirement for this defined. In terms of SW Engineering, this is a common issue where the requirements need to be revisited due to a faulty or not-good-enough design.

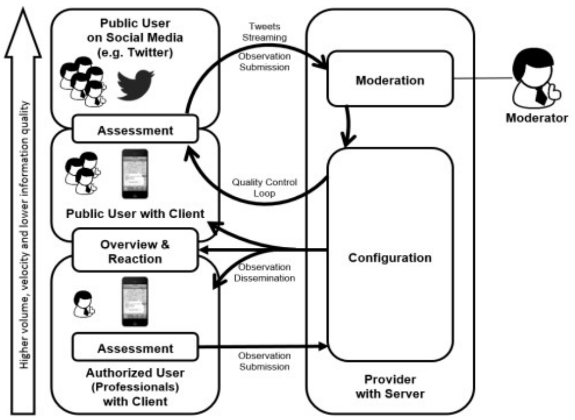

Figure 3: **GDACSmobile Information Flow Model**

Testing and Integration of the system was extensive, as the writers had a predefined set of data that this system would be trained and tested on (natural disasters). Other than the challenges outlined in the paragraph above, there were no additional challenges defined. However, in the larger context of SW Engineering, on challenge may be that the test data is not wholly encompassing of all test cases. It is also hard to define test cases for systems of social context.

Finally, the Implementation and deployment (maintenance) stage of SW development was less technical. In the study, the writers talk about probable ways to deploy this system but were turned down by larger social media enterprises. The study says it is because the system is not "trusted" to be effective by potential users. In the broad context of SW Engineering, this is equivalent to the user not being satisfied with the system and the project needing another iteration before evaluation.

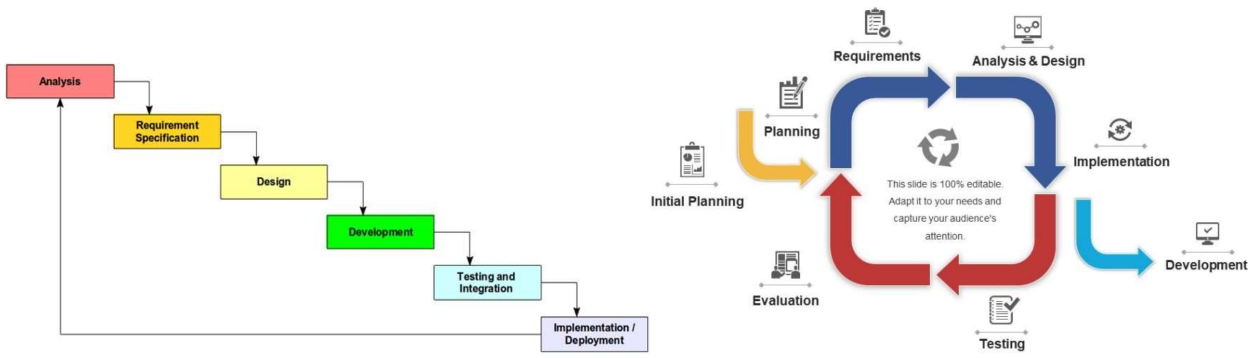

Figure 3: **Waterfall Model [3] and Iterative Process Model [4]**

Overall, the system seems to follow the Waterfall Methodology, while simultaneously running into common issues of software engineering, causing some backtracks in the model to the previous step, which later was a driving factor in requiring the project to go into another iteration of development.

4 Areas of Concern and "Best Practices" for Software Development

By using the case study as a basis, we found five areas of concern and five best practices for software engineering for automation development. These are listed and explained in the subsections below.

Areas of Concern:

1. *Time and Cost*

Time is money and having multiple iterations is extremely expensive in both aspects. The study was only able to create the first iteration of the prototype, but spent more resources for comprehensive testing that would greatly benefit a second prototype. The biggest idea here is what phase to spend your time and money in the project and to be able to justify the costs with data.

2. *Good Human-Computer Interaction (HCI)*

HCI is much harder to implement than Computer-Computer Interaction since either entity may not understand the other – there are a multitude of steps needed to facilitate proper communication. In terms of a worst-case scenario for this case study, the system could be used to filter information about natural disasters to allocate humanitarian aid. The HCI could result in a life-or-death scenario that allocates resources to the wrong area at the wrong time, leading to people in high-risk situations dying.

3. *Ease of use*

In addition to the former point, during development, software usage can be perceived differently between the user and the designer, causing the user to not understand how the program works, which would result in a failed iteration. This also ties into the point below.

4. *Customer Preferences*

Customers may not always be specific in terms of how they want applications to look and feel. This plays a large part in the "acceptance testing" for the system. Additionally, much of the customer's feedback may be highly subjective, making it hard to interpret for the engineer. For instance, in the case study, potential customers did not like the testing of the system, disregarding it as untrusted, despite not giving any sample verified data or any feedback, making it extremely hard to build another iteration.

5. *Future Feasibility*

It is entirely impractical to create a piece of software that will be replaced before it can generate profits. If the system is not reusable – that is, it only works on the parent system that it was built on, the software will become redundant quicker than intended. In terms of the study, GDACSmobile was a tool used that could port the system elsewhere. The system also took a non-existing approach to content moderation that could benefit its potential users more, which gives the software value.

Best Practices:

1. Periodic customer feedback

In SW Engineering, it is important to maintain periodic contact with the customer to ensure that all part of the system are developing to customer standards. For instance, customers can be used to procedures for certain systems, like mentioned in the case study, which would require the engineer to design their interface a similar way. This practice is even more important for HCI systems since the computer an only take a defined set of inputs and the user may be confused. The customer may also have concerns about security and ease of access, or other functional requirements for the system that are not outlined initially.

2. Defensive designing

Defensive designs are a good way to reduce and handle errors in a system. During development, it is important to note any potential errors and have a handler designated for said error. That is, we do not trust the user to never make a mistake and assume an error may occur. In terms of an example from lectures, an error could be of an ethical standard. If a self-driving car were to choose between an old lady and a child to avoid hitting, the solution may have to be preprogrammed in.

3. Good, in-depth iterations

Most automated systems will go through multiple iterations before deployment. Between iterations, it is important to go through a full iteration of development and get as much information as possible at each stage before moving on. For instance, the case study mentions how the first iteration of the semi-automated moderating system lacked features like relevance scoring, feature extraction, etc. Feedback like this is extremely important to maximize the yield of later iterations and decrease the costs of the system.

4. Using Validated Test Data and Tools

One of the most important topics explained in the case study was how the study proved the relevance and benefits of the tools it used for developing the automated system, and whether or not the tools would be a valid match for the project.

5. Good Code Reusability and Integration

When defining the requirements for the software, it is beneficial to identify a *problem set* that is solvable with the automated system, or opportunities for integrating the system. For instance, the study focuses on automating content moderation through a filtering mechanism that can be used in information retrieval or other enterprise application frameworks.

5 TOOLS FOR SOFTWARE AUTOMATION DEVELOPMENT

In a broader view, many software engineers use tools to automate the software development process. However, this can be very risky, especially since software generally will not be written in human-readable format. That is, SW Automation Development tools need to strengthen or ease the development process. The study uses the GDACSmobile system for instance. To automate development, one tool used is hidden markovian modeling with clustering analysis to determine viable tools to use in a system, which would semi-automate the development stage. [5] uses this idea to classify malware, but some tweaks to the system to compare development tools' performance to known, validated systems, to determine the best tools for implementing the system. One issue with using hidden markovian models is that some non-functional requirements like ease of use and performance are lacking. Also, depending on the system, the scores may not be relevant unless there is a comprehensive trainer used for train the models and create accurate clusters/scores. However, one limitation of the system and the development process is that future processes are only as strong as the preceding process, which is important in avoiding a "software crisis".

6 CONCLUSION

In conclusion, this research report summarizes a case study on automation content moderation and reports specific challenges in the development process of the case study's system. This report also outlines five areas of concern and best practices for developing software, giving examples in how it relates to the case study and the broader scope of software development. Finally, this report analyzes and discusses the use of markovian modeling in automating software development.

7 References

[1] Link, D., Hellingrath, B. and Ling, J. (2016). *A Human-is-the-Loop Approach for Semi-Automated Content Moderation*. [online] ISCRAM 2016 Conference. Available at: https://pdfs.semanticscholar.org/2223/f7245f7c310db2c4a24e6e4603c85936d460.pdf [Accessed 23 Jul. 2019].

[2] Weber, S. (2010). *Design Science Research: Paradigm or Approach?*. [online] Sixteenth Americas Conference on Information Systems. Available at: https://pdfs.semanticscholar.org/7c31/a4019ab19db0123c90ba72250bc99d80ace5.pdf [Accessed 9 Aug. 2019].

[3] Gladkova, E. (2012). *Agile vs Waterfall Development model*. [online] SysGears. Available at: https://sysgears.com/articles/agile-vs-waterfall-development-model/ [Accessed 8 Aug. 2019].

[4] SlideTeam. (2019). *Iterative Process Diagram*. [online] Available at: https://www.slideteam.net/iterative-process-diagram.html [Accessed 8 Aug. 2019].

[5] Annachhatre, C., Austin, T. and Stamp, M. (2014). Hidden Markov models for malware classification. *Journal of Computer Virology and Hacking Techniques*, 11(2), pp.59-73.