Delft University of Technology

Software Project

CSE2000

AWSome NLP

Low Resource Neural Translation of AWS Blog Posts

Authors:
Group 7C
K. Silvester Caldera 5218225
Athanasios Christopoulos 5318157
P. Olivier Lacombe 5207029
I. Singh Pahwa 5509858
Lara Sakarya 5262712

Supervisors:
Halil Bahadir
Esra Kayabali
Hasan Basri Akirmak
Anastasia Pachni Tsitiridou

June 2, 2023

Delft, Netherlands

Contents

Acronyms

LHS Left Hand Side. 6

LRL Low Resource Language. 2, 3, 5, 7

NLP Natural Language Processing. 3, 7

RHS Right Hand Side. 6

UI User Interface. 9

Glossary

Amazon CloudWatch Amazon CloudWatch is a monitoring and management service that provides data and actionable insights for AWS, hybrid, and on-premises applications and infrastructure resources. 8

Amazon SageMaker Amazon SageMaker is a cloud machine-learning platform that enables developers to create, train, and deploy machine-learning models in the cloud. 3

Amazon Translate Amazon Translate is a neural machine translation service that delivers fast, high-quality, affordable, and customizable language translation. 3

Low-resource language Low-resource languages (LRLs) are those that have relatively less data available for training conversational AI systems. 3

LSTMS long short-term memory networks, it is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems. 4

MoSCoW The MoSCoW method is a four-step approach to prioritizing project requirements, it stands for must have, should have, could have and won't have. 6

Natural Language Processing Natural language processing (NLP) is a machine learning technology that gives computers the ability to interpret, manipulate, and comprehend human language. 6

Neural Translation Model A NTM is a model based on neural machine translation, which is an approach to machine translation that uses an artificial neural network to predict the likelihood of a sequence of words, typically modeling entire sentences in a single integrated model. 6

Preface

preface text here...

Summary

Summary text here...

1 Introduction

An estimated 80% of the internet's content is available in only 10 of the identified 7000 languages [1][2]. This means that the world's population has limited to no access to digitally translated content in their native tongues. Low-Resource Languages (LRLs) are often marginalized in the digital space and, as a result, the native-speakers of these languages have limited access to tools and opportunities on the Internet. For the Software Project, we are working for Amazon Web Services (AWS). AWS is a subsidiary of Amazon.com, Inc. that offers cloud services such as computing power, storage for databases, and other technologies that provide reliable, scalable, and cost-optimizable services. As part of their infrastructure, the client provides valuable AWS Blogs showing its users valuable information related to their cloud-computing infrastructure. AWS wants to make this information available to all and has therefore asked us to design an application which takes an AWS blog and is able to translate it to a target LRL. This report aims to present the research conducted and implementation process of designing and developing a full-stack application capable of translating AWS Blog Posts into an LRL using either Amazon Translate or a neural translator designed by the group. The purpose of this research was not just to devise a solution, but to make AWS blog content more accessible and inclusive, thereby bridging the linguistic divide.

The primary aim of our research is to answer the question: Can we create a full-stack application that translates AWS Blog Posts into a low-resource-language effectively using Amazon Translate or a custom in-house model built and trained on Amazon SageMaker? This report will discuss the process of implementing all parts of the application as well as training our model to translate English AWS blog posts into Turkish (a LRL) in the specific AWS tech domain.

This report has been structured to provide a comprehensive understanding of our research and its outcomes. It begins in Chapters 1 and 2 with a detailed overview of the theoretical underpinnings of machine translation and the challenges associated with translating LRLs. Chapters 3 and 4 explores the research identifying key resources and limiting conditions such as software and data limitations. This is followed in Chapters 5 and 6 by a description and implementation of our methods, including all sub-level concepts and explanation of the different AWS services used in development of the application. Subsequently, in Chapter 7 and 8 discuss the testing and evaluation of our final product, effectively comparing translations generated by Amazon Translate and those by our own model. The final sections cover the implications of our research, potential future work, and our conclusions. We then explore the ethical and societal implications of our work in Chapter 9 before concluding our findings in Chapter 10.

2 Problem Analysis

AWS has an existing blog with thousands of blog posts [3] written by employees and researchers covering existing AWS technologies/services. These blog posts describe a variety of media types including news, technological guides, and reviews on the extensive technologies provided by AWS. As a part of the AWS infrastructure, it is an instrumental piece in how it interacts with its community: by keeping them up to date on new developments and helping them understand how to use their technologies.

Unfortunately the vast majority of these blog posts are predominantly written in English, a language only native to 5% of the world population and spoken as a primary or secondary language by 17% [4]. While 12 other *editions* of the AWS blog posts exist, in languages like French, Thai, or Korean (among others), they may contain language-specific content not accessible across *editions*. This creates a language barrier for users not speaking the language, meaning non-speakers do not have access to understanding the content and resources provided on these posts.

While modern software solutions do exist for translating text; there is no easy, clear-cut means for translating these posts in their entirety. Users who want to do this may have to resort to third parties or translate pieces of the blog section by section. This issue is further exacerbated in the scope of low-resource languages where existing solutions may provide a limited or inadequate translation, especially given the technical nature of the blog posts. This is problematic as these valuable AWS blog posts **should** be accessible to everyone and not present any barriers on the basis of language.

This section describe our problem analysis, beginning with the problem statement (1.1) and ending with topic research (1.2) with respect to each stakeholder, use case, and existing technology.

2.1 Problem Statement

AWS has a blog with thousands of blog posts, all containing unique and valuable information, spread across 12 different *editions*; all demarcated, divided, and separated by language. Some of these articles and languages are easy to use for translation, given current machine translation tools, however, others, specifically Low Resource Language (LRL)s, are not. This means that speakers of certain languages do not have access to valuable information due to a language barrier. This information hinders students, researchers, developers, and employees alike. Our task is to create a tool that reduces the impact of a language barrier and allows these blog posts to be translated from English to a Low Resource Language (LRL), namely, Turkish while taking into account that this project should be scalable and extendable to other low-resource languages.

2.2 Topic Research

2.2.1 Stakeholders

The stakeholders of this project can be distinguished into two groups: those that are impacted by **using** the service and those that are impacted by the **use of** the service. Of the groups that directly use the service, there are two distinct subgroups: Turkish speakers and non-Turkish speakers. The Turkish speakers will be discussed together with the group that is impacted by the **use of** the service, as they are more similar.

Users

The first group of stakeholders that will be considered are the people that **use** the service, otherwise known as users. As AWS already has an infrastructure for blog posts accessible in foreign languages, the group of non-Turkish speakers that wish to use the service will often be contributing to the foreign language *editions* of AWS blog posts. These people include researchers that work with and/or for AWS, developers that have notable findings, blog writers that aim to reach a certain or wider audience, and executives that are publishing regular reports. Ideally, the service would allow a seamless translation of a blog post from one language to the other, allowing foreign language readers to have immediate access to the information on their *edition* of AWS blog posts. The users of this service would then experience greater productivity in translating the paper themselves, as opposed to finding a translator that has the necessary domain knowledge, source language knowledge, and translated language necessary to execute the translation [5]. Furthermore, the existence of the service may give users a convenient way to translate their posts, reaching an audience that couldn't be reached before.

Other stakeholders

Conversely related to the stakeholders using the service, are the stakeholders being impacted by the **use of** the service. This group will include students, researchers, and executives wishing to access information in a language they are comfortable in reading, professional bilingual translators, and readers of *pre-translated* blog posts. The first group, students, researchers, and executives, consist of a group that may have some knowledge of the source language, perhaps just enough to comfortably understand the main idea of the blog post, but, not enough to completely comprehend and digest the blog. This group is the aforementioned group that is technically in the users' group but is more similar to the native language *users* in that this will grant them access to information that was not previously accessible to them.

The final group of stakeholders is professional bilingual translators. It has been shown that monolingual translators aided by a machine translator performed better than bilingual translators [5]. These professional bilingual translators are certainly to be impacted by this service (at scale), as they will either be aided by its existence, and made more productive, or be replaced by the machine translator and domain knowledge of the monolingual translator, with the latter seeming more likely.

2.2.2 Use Cases

The intended use case for this project involves translating AWS blog posts from a source to a destination language. This would allow interested users who don't speak the source language of an AWS Blogpost to access this content in their native tongue. Although this project is **intended** to translate from English to Turkish and only to be used for AWS blog posts, a translation tool for written text in a Low Resource Language is widespread. The NLP aspect of this tool can be extended to receive input and give output through various forms and in other (non) AWS domains.

This project has the additional use case of being able to compare translation methods. AWS already has an existing team of translators who have written certain English texts in Turkish. This team, alongside the existing Amazon Translate technology and the in-house neural translator built by the team will allow AWS (and interested users) to compare these different technologies.

2.2.3 Existing Technologies

There is a wide range of technologies in the domain of machine translation systems. There has been an emergence of software and research in NLP translation [6] and low-resource language translation [7]. Perhaps the largest and most popular machine translation software for full website translation is Google Translate, a neural machine translation software capable of translating entire webpages [8]. However this technology, among others, is a direct competitor to the client's Amazon Translate and doesn't manage to solve the problem in its entirety. They are not targeted to dealing with AWS blog posts specifically and have a tendency to under-provide and under-perform on low-resource languages, especially for different dialects [9].

Fortunately, we are asked to incorporate Amazon Translate, a powerful translation tool as part of our solution. In addition to Amazon Translate, we are asked to use Amazon SageMaker, a Jupyter Notebooks style tool that allows us to build, fine-tune and host a transformer-based language model. This means we will incorporate the existing technology (In Amazon Translate) and develop our own through SageMaker with the aim of solving the problem. We can use heavily researched and readily available knowledge in Natural Language Processing (NLP) [6] and Low-resource language translation [7] [9] for approaching the task at hand.

2.2.4 User and Expert requirements

Fortunately we have easy access to experts to consult in the development of this software project. This project is provided and supervised by Halil BAHADIR, Esra Kayabalı , and Anastasia Pachni Tsitiridou who are all experts in designing and implementing tech solutions (AWS Solutions Architects). They can provide valuable information on what is wanted and needed in the project. These precise requirements will be further discussed in section 4.

Our access to users is a little more difficult. Our solution is aimed at translating *English into Turkish* and as of now we do not have access to users who speak Turkish and **do not** speak English. However, one of our group members (and some of our project providers) are native Turkish Speakers and could be seen as *pseudo-users.*

2.3 Project Goals

3 Feasibility Study & Risk Analysis

When considering the execution of a project, its feasibility and possible risks are tantamount to the design and engineering itself. For this reason, this section will cover, first, a feasibility study about the scope and context of the project (2.1), covering both technical and non-technical feasibility, followed by a risk analysis (2.2).

3.1 Feasibility Study

When considering technical feasibility, the scope and context of our project are vital. Creating a full-stack web application with a translation tool in 10 weeks seems to be feasible, albeit somewhat challenging. We are very comfortable working with full-stack applications since we have had relevant courses (Object Oriented Programming Project, Software Engineering Methods, and Web & Database Technology) that extensively cover the development of full-stack applications. The creation of a ML model that accurately translates a LRL seems to be the part of the project that takes the most time and effort. While we have done courses (namely Machine Learning, Data Mining, and Computational Intelligence) whose curriculum includes a lot of the theory behind the basics of the implementation of these technologies, there are still various technologies, services, and theories that we haven't covered in our courses so far (for example, Transformer based models and LSTMS). Research into these further topics will be key, however, we have access to AWS services, which drastically cuts down on the amount of time needed on creating a viable ML model, and all the specifics with a full-stack web application, while still allowing for scalability and continuous deployment. Without going into too much detail on the specific services we intend to use, information on this will be provided in Project Approach and Development Methodology sections.

Technical Feasibility: SWOT Analysis

The following analysis will be conducted regarding the strategic planning of the project on its strengths, weaknesses, opportunities, and threats to evaluate the planning and design of the project's solution:

- Strengths: The most significant strength of this project is the availability of AWS technologies provided by AWS. The following services described will be used to build the full-stack web application: AWS SageMaker and AWS Amplify among other services which may be of use in our application. The use of these services will optimize the cost as well as the time spent on certain tasks. [**?**].

- Weaknesses: The dependency on the AWS services could limit the design choices for the web application due to the limited customization options. Due to working with a low-resource language (Turkish), there could be limited training data available to train the ML algorithms which may affect the reliability and quality of the translation. Due to limited resources, the actual meaning of sentences from Amazon blog posts may not be captured accurately, leading to an underperforming and under-providing translation [**?**].

- Opportunities: The translation of the blog posts can have a significant outcome in terms of expanding AWS's market since the use of AWS services may increase due to the increase in the availability of technical information on AWS to Turkish users. AWS's business can perform better in the regions with Native Turkish Speakers. After the completion of the web application, the process of translating other low-resource languages will be made easier, hence allowing AWS to expand its business with new opportunities. Lastly, the use of AWS services throughout this project will enhance the developers' understanding of ML and cloud computing while allowing them to improve their technical skills and benefit from these services in the implementation of their web application [**?**].

- Threats: Other translation services such as Google Translate could be used instead of our translation service since customers could find them more reliable due to their long-term existence and use. Due to the low-resource nature of the language, the training data may not be enough to build an accurate model resulting in translation mistakes which could reduce the reliability and use of the application. The level of experience of the developers of this project could concern the customers on the quality of the translation service, leading to fewer customers using it. The use of AWS and the personal accounts could potentially pose a problem in the case of misuse which may result in financial damage to AWS [**?**].

  There are various different aspects of this project which need to be considered while implementing the web application, especially from a security, reliability, and maintenance approach.

Non-Technical Feasibility

Besides the analysis regarding technical feasibility, there are also certain things to consider on the non-technical side of this project. Technical support is a significant part since there will be AWS experts with high availability to consult on the progress of the project, especially for guidance with the services. The impact of this application on society should also be considered since this project will highly affect users of AWS, specifically Native Turkish Speakers, that couldn't have high access to information on AWS before. The understanding of low-resource languages in terms of their definition as well as what hinders them from becoming high-resource should be well thought out to support the development process of the ML model. Turkish as a language should be analyzed for its difficulties that could possibly pose certain risks to the accuracy of the translation service.

3.2 Risk Analysis

It is paramount to consider all potential factors that may hinder, or outright prevent, the final delivery of the project. For this reason, we have considered a set of possible risks and their danger of interfering with our ability to meet the client's needs. Some of these risks are simply outside of our control while others are reasonably in our hands, and can therefore be mitigated.

In the scope of this project, we have a hard dependency on using the client's AWS services. This is an integral part of our project and failure to do so is considered a failure to complete the project, however, this dependency poses a potentially low risk as AWS is considered to be incredibly reliable [10]. Additionally, we have a risk concerning the quality of the data for creating our translation tool (through SageMaker). If the client is unable to provide us with sufficient quality data to train the model, this core requirement is unfeasible. This is considered a *medium-level risk* because we are translating **to** an LRL and the AWS Turkish Blog Data-set in it's entirety may not be enough to train a full-blown translation model.

The team needs to develop a scalable, well-tested, full-stack application utilizing unfamiliar AWS services within 10 weeks while only being able to meet the client in-person weekly. Inadequately planning the project, time management, collaboration, or motivation may lead to a poor final product or failure altogether. Being evaluated at a *low-medium level risk* means the team will not only need to use its resources wisely but also ensure all tasks are properly planned and completed successfully in a timely manner.

4 Requirements

Requirements are an essential part of project planning. It is a way to ensure the joint comprehension of the client and developers of the expected functionality of the deliverable. We have created a list of the requirements we identified using the MoSCoW model. Given that we worked closely with the client, we believed this form of collaborative (between client and stakeholders) requirement elicitation through the MoSCoW method would most benefit this project [11]. Following the MoSCoW model, we divided our requirements into functional and non-functional categories.

4.1 Functional Requirements:

Must Have

- Translation: The application must have the following translation requirements
  - The application must translate AWS blog posts into a low-resource language (Turkish).
  - If the blog post contains code, translation must be avoided and code should remain as is.
  - Title of the blog post must be translated.
  - User must be able to pick between 2 translation modes.
    1. Amazon Translate
    2. Neural Translation Model built off Amazon SageMaker (This is technically a must have, however, due to the nature of training Natural Language Processing machine learning models, it will not be ready by the MVP deadline)
- Input: The user must be able to enter a URL.
- Output: The user must be able to view the translated document.

Should Have

- Input: The following functionality should be available when inputting a request.
  - Technical jargon should not be translated.
  - Application should allow the user to search/toggle between output language(s).
  - Application should allow the user to toggle between translation techniques.
  - Application should only allow AWS blog links to be pasted.
- Output: The user should get the original post on LHS and translated post on RHS.
  - Posts should be scrollable and have elements aligned on both sides.
- Interaction: The application should allow for interactive functionality on the output:
  - User should be able to highlight parts of the input (and reflect on the output).

Could Have

- Input: The following functionalities could be added when inputting requests.
  - Application could allow for the user to 'search' for blogs by title if there is metadata on them.
  - Application could automatically infer the blog's language (Assuming translating from more languages than just English).
- Output: The functionalities of the website (hyperlinks and share) could be provided in the translation.
- Interaction: The following interactive functions could be implemented.
  - User could export the blog with its translation and print it.
  - User could be able to switch between translation modes seamlessly.

    1. If currently viewing Amazon Translate translation, the user could seamlessly switch to NLP translation (and vice versa) with a button, etc.
    2. Same positions of elements could remain on both sides.
  - Application could have a way for users to give star ratings on how well the blogpost was translated
  - Highlighting/editing sections could be seamless on both sides (no need to re-translate).

- Authentication: The application could have a form of authentication with the following.
  - Application could allow users to register (using social media SSO's).
  - Application could allow users to identify themselves to view/save previous notes/translations/edits.

- Translation: The blog post's author and information about them could be translated.

Won't Have

- Translation:
  - Translation won't have support for multiple languages and will only be one way (English to Turkish).
  - The comments of blog users won't be translated.

- Interaction: The user won't be able to edit the blogs or the translated version of the blogs.

4.2 Non-Functional Requirements:

Must have

- Performance: The application must have a fast response time on all functionalities.
- Reliability: Application must have limited to no downtime or disruptions.
- Compatibility: The application must be accessible across popular browsers/operating systems on a desktop.
- Usability: The application must be easy to use. Users intuitively know how to do all functionalities.
- Translation: The translations provided by the NLP are accurate and natural to target LRL (Turkish).
- Technology: AWS services will be used, in particular, Amazon Translate for the translation and Amazon SageMaker to fine-tune the ML model. An AWS service of choice must also be used to deploy and host the application.
- Data Privacy: User data must be well protected, adhering to regulations. (GDPR).
- Re-usability: The project must be built in a way that ensures future developers can easily continue development and feature integration.
- Scalability: The Application must be easily scalable to add new functionalities.
  - Application must be able to handle multiple users at a time and should scale to handle upto 100 users.
  - Software must allow easy integration of translation to other LRLs.

Should Have

- Documentation: Codebase should be well documented using a coherent format throughout.
- Visually: The application should be aesthetically pleasing and similar to other AWS websites.
- Accessibility: The application should be accessible for people with disabilities following WCAG2.1
- Testing: The application should cover the following testing.
  - unit testing

  - integration testing
  - functional testing
  - acceptance testing

- Technology: The following technology should be used.
  - An AWS service should be used to store data (like DynamoDB). This database should be for blog-posts, though the concrete use may vary.
  - Java as the main language for the project.
  - Python as the secondary language primarily used for NLP and any other possible AI/machine learning functionalities.
  - A CI/CD pipeline should be used in the application and in the infrastructure of the code.

- Security: The application should defend against common cyber attacks.
  - DDOS attack security
  - Injections (if we are using a database)

- Scalability: Software should allow easy integration of new translation methods. (Beyond Amazon Translate and NLP).

Could Have

- Translation: The translations provided by the NLP are capable of translating technical jargon in a way understandable by LRL
- Monitoring: The application could have a form of monitoring aspects like performance and reliability.
  - Monitoring performance and reliability could include: checking the compiler, execution, invocation, duration, error comments, success rate, and connecting to Amazon CloudWatch.
  - Alerts could notify developers of any failures in the system.
- Authentication: The application could have the option for users to pay for additional services which include logging in and saving/downloading the translations.

Won't Have

- Accessibility: The application will have the following accessibility restrictions.
  - The application won't have a mobile version
  - The application won't be downloadable (web application only).
- Usability: The application won't have instructions that guide users through the process.

5 Project Approach

In this section, we outline the key decisions outlined in designing our application. The detailed aspects of our software architecture, technology choices, and design choices will be further discussed. This was an integral step in the initial planning on the project and was still re-visited throughout the project given the architectural complexity and novelty of the technologies used.

5.1 Software Architecture

5.2 Technology Choices

With the exception of AWS SageMaker and AWS Translate, the client is very lenient with which of the Amazon services we use. AWS SageMaker plays an important role in our application for developing our translation service using machine learning. It provides a platform to easily apply ML models and is scalable in the building and training process, offering efficiency and cost optimization. With the current knowledge of the group, Figure 1 shows our current approach to the project's development though many of these services may change. This is due to our weekly meetings with the client and discussions regarding our progress as well as consulting them in case we need assistance with any of their services. While our exact methodology will be addressed in Section 6, the client has explicitly stated that our approach should allow for feedback and opinions (from the client and development team) and maintainability such that we can have a *flexible* approach. This flexibility means we can re-evaluate and deviate from the original project plan throughout development and perhaps use alternative services or implementations.

Flexibility in mind, our high-level approach and requirements were stagnant. There is a plethora of AWS services we can use so as a result, In the initial stages of the project, we intend to familiarize ourselves with the AWS platform and some of its various services to integrate them into our development and enhance our understanding of the project as well as the client's requests. Once familiarized with the AWS ecosystem,through research and use of these services, we felt more comfortable beginning the implementation process.

In developing the implementation our initial plan was to start building the application in the shape of a working Front-end, Back-end, and communication between the two. Then, we intend to directly cover the functional requirements presented in Section 4 in order of importance, while still keeping non-functional requirements in mind.

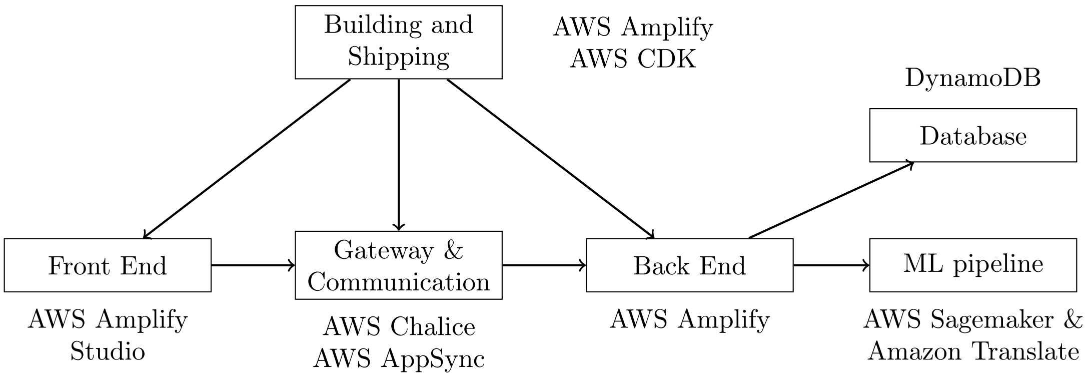
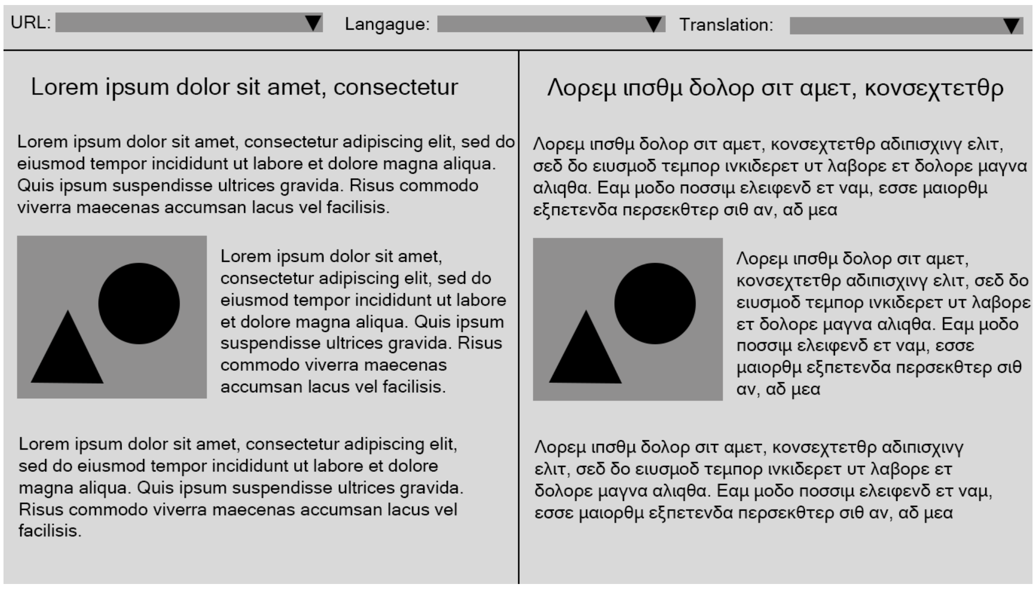


Figure 1: Use of AWS services in development

5.3 Design Choices

Figure 2 presents the UI the group has envisioned for the project. We intend for the application to roughly follow this design where we see the original text on the left and translated side on the right. We also intend to give the user the ability to easily translate a different blog post (URL), change the translation language (Language), and the method of translation (Translation). As part of our design, we intend for translations and objects to appear side by side, providing a clear alignment between the original and translated blog.

This initial design is predominantly a product of the client's exact wishes. Claiming that they wanted these exact features visually displayed on the web application. That said, it still follows common design heuristics which are significant to build a bridge between the user and the application to achieve a certain goal [12]. The design is simplistic, efficient, and the user's has been taken into account, meaning the UI/UX allows for usability of the application on the basis of the user's needs while prioritizing user satisfaction through this feedback cycle [12]. The accessibility of the application is another important aspect since users with disabilities should be able to use the application as easily as any other user, meaning the final product will follow general WCAG2.1 guidelines.



Lorem ipsum dolor sit amet, consectetur

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Quis ipsum suspendisse ultrices gravida. Risus commodo viverra maecenas accumsan lacus vel facilisis.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Quis ipsum suspendisse ultrices gravida. Risus commodo viverra maecenas accumsan lacus vel facilisis.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Quis ipsum suspendisse ultrices gravida. Risus commodo viverra maecenas accumsan lacus vel facilisis.

Λορεμ ιπσθμ δολορ σιτ αμετ, κονσεχτετθρ

Λορεμ ιπσθμ δολορ σιτ αμετ, κονσεχτετθρ αδιπισχινγ ελιτ, σεδ δο ειυσμοδ τεμπορ ινκιδερετ υτ λαβορε ετ δολορε μαγνα αλιqθα. Εαμ μοδο ποσσιμ ελειφενδ ετ ναμ, εσσε μαιορθμ εξπετενδα περσεκθτερ σιθ αν, αδ μεα

Λορεμ ιπσθμ δολορ σιτ αμετ, κονσεχτετθρ αδιπισχινγ ελιτ, σεδ δο ειυσμοδ τεμπορ ινκιδερετ υτ λαβορε ετ δολορε μαγνα αλιqθα. Εαμ μοδο ποσσιμ ελειφενδ ετ ναμ, εσσε μαιορθμ εξπετενδα περσεκθτερ σιθ αν, αδ μεα

Λορεμ ιπσθμ δολορ σιτ αμετ, κονσεχτετθρ αδιπισχινγ ελιτ, σεδ δο ειυσμοδ τεμπορ ινκιδερετ υτ λαβορε ετ δολορε μαγνα αλιqθα. Εαμ μοδο ποσσιμ ελειφενδ ετ ναμ, εσσε μαιορθμ εξπετενδα περσεκθτερ σιθ αν, αδ μεα

Figure 2: Visual Design of Application

6 Development Methodology

6.1 Project Methodology

6.1.1 Our Findings

We have conducted research on four popular development methodologies: Agile, Waterfall, Spiral, and Vee. More of our findings and reasoning behind our choices are in the Appendix, however, for brevity, we have include a bulleted list of the pros and cons of each of these methods.

- Agile
  - Focus on productivity, flexibility, and practicality [13]
  - Lack of documentation and formal studies
- Waterfall
  - Document driven, simple, clear-cut milestones [14]
  - Inflexible [14]
- Spiral
  - Iterative planning approach for risk-reduction, with continuously increasing detail [14]
  - Heavy-weight, time-consuming
- Vee
  - Explicit focus on verification and validation; a lot of testing in mind [15]
  - Not very little in the way of implementation, and inflexible [14] [15]

6.1.2 Our Methodology of Choice

All of the aforementioned models have takeaways that we would like to incorporate in our methodology: iterative refinement of requirements of the Spiral Model, the emphasis on verification and validation from the Vee Model, the explicit documentation and demarcation from the Waterfall method, and the flexibility and iterative implementation approach of Agile. Ideally, we'd like all of these aspects from each of the different frameworks, however, we would not be able to get all of them from simply following any one of them verbatim, so we feel the need to engineer our own hybrid approach, that answers Boehm's questions for process methods: "What shall we do next?" and "How long shall we continue to do it?" [14, p. 61].

6.1.3 The Chalice Method

Trying our best to incorporate all of the aspects from the previous methodologies, and what we discussed with industry experts, we came up with the chalice method, which takes inspiration from all of the aforementioned methods and what AWS practitioners actually use. The area under the blue dashed lines is the *Scrum*; this will occur on a regular basis, very similar to a standard Agile method of implementation. This will keep looping until the team is ready to complete the project. The dotted lines represent back-edges that, ideally, should not be followed, but could be if the project calls for it. Stages outside the *Scrum* subsection are complete when a formal document depicting them is complete, much like the Waterfall method. Stages within the *Scrum* subsection are time-based, with a sprint occurring once a week, much like standard Agile, however, alongside the standard completion criteria that come with Agile, the Chalice method adds a condition, namely, the Software check/ review. For our project, we finish at the Operating and Maintenance stage.

Top Level Concept (TLC) For our project, this part is rather straightforward, it is the grand-scheme concept of what we're building. In this stage, we got introduced to the client, and the project, and selected our low-resource language.

Requirements Similar to the Vee method, this is the stage in which we develop our requirements, and as such, our initial architecture. Based on our highest-level concept, we are able to develop requirements and user stories.

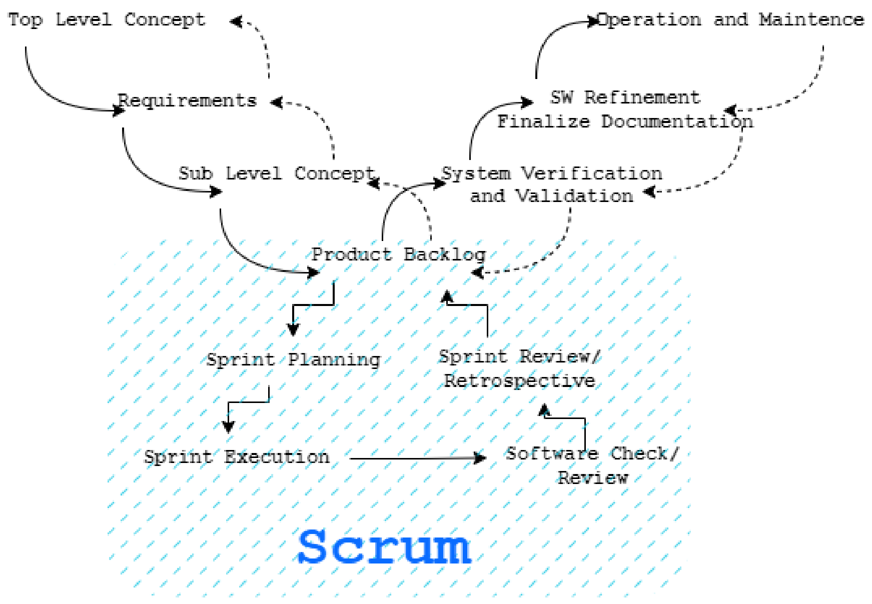


Figure 3: The Chalice Method: Alternative to popular methodologies

Sub-Level Concept (SLC) Using the requirements, the team is able to come up with a more detailed architecture, which for this project, will give us a solid, detailed foundation and help us select and pinpoint which AWS services we'd really like to use and focus on. Furthermore in this stage, we'd solidify our requirements and technologies, which would lead to a very straightforward implementation approach, since, ideally, all of the architectural kinks should be ironed out during this stage.

Product Backlog This is the first and last stage of the *Scrum* subsection of the model. This replaces the "Implementation" step of the Vee model, giving the team a more rigid framework to actually implement and create the software, based on Agile. Unlike the Vee model, this model is not entirely dependent on previous documentation and different levels of conceptualization to implement the product, instead, this model allows the different concepts to pave the way for the engineers, and create structure, while still allowing them to move freely and make changes within their sprints. This stage has the team examining the product backlog, creating, modifying, and removing user stories and requirements ad-hoc, but at fixed points in the cycle.

Sprint Planning and Sprint Execution These steps are identical to the normal Scrum methodology, wherein our Sprint Plannings are to occur once a week, and our execution is to take place by daily stand-up.

Software Check/Review Based on the conversations we have had with industry experts, we believe, within the Agile-inspired subsection of the Chalice Method, we needed a mechanism specifically designed for receiving feedback and creating documentation, which is where Agile falls short. Our solution to this problem was to modify code reviews frequently found in Agile by also placing emphasis on documentation and code clarity. We perform these modified code reviews at the end of a sprint. Software that meets requirements **must** pass this step before being complete.

System Verification and Validation After a sufficient number of sprints have been completed, the project can move onto the right wall of the chalice, which begins, similar to the Vee method, with the System Verification and Validation stage, where the team ensures that the system as whole works, using various testing methods, and that it meets the requirements.

SW Refinement and Finalize Documentation In this stage, the team ensures that the software is organized and finalizes formal as well as in-line documentation to prepare for the hand-off. If SW refinement occurs, the team may need to take a step back to the previous step.

Societal Relevance and Ethical Approach sIt is not designated in the diagram, however, during the development process of our translation service, we will critically think about the impacts our application could have on society, especially on Native Turkish Speakers. The possible concerns which may arise due to the quality of the application will be kept in mind throughout the development process, and the ethical side of our application will be thought out to make sure we follow the best practices without any negative affect on society.

6.2 Communication & Tools

As a team, we decided that we will use mainly 3 channels of communication, Discord for online meetings, WhatsApp for general communication, and Mattermost for communication with TAs and the course staff. We use Discord for general meetings related to the project and our daily stand-up sessions where each one of us briefs the entire team on what we, as individuals, have been working on the previous day, what we hope to achieve in the future and ask for clarification or give useful information related to the project and our schedule throughout the meeting. We also have a shared Google Calendar that helps keep us on the same page with regard to the schedule as it is easier than coordinating meetings/deadlines over text communication since it is easy to forget or miss a message. We are also interested in meeting in person, either at AWS' offices or on campus as we find that we work best as a group when we are all in the same location but we are not sure if working at AWS offices is feasible yet. Furthermore, we use AWS' internal service, Amazon Chime, to have online meetings with AWS employees. We meet with the technical team at AWS every week on Fridays to discuss our progress and to ask any questions we might have. Furthermore, we will have Gitlab's scrumboard to keep track of our issues as we go .

7 Design & Sub Level Concepts

The application aims to combine eight different AWS services. There include AWS Amplify, AWS AppSync, AWS Lambda, AWS Step Functions, Amazon Translate, AWS SageMaker, Amazon S3, and Amazon DynamoDB. The use of each of these services, and their use in building our application, will be discussed in this section. Overall, in an architecturally complex application with a vast amount of moving parts, the optimal and efficient use of the different AWS service will allow us to have a scalable, accurate and reliable full-stack web application for translating Amazon blog posts.

7.1 Development and Communication Services

AWS Amplify is introduced as a development platform for building web applications. AWS Amplify simplifies the development process and facilitates integration with other AWS services, such as Amazon S3 and AWS AppSync. Amplify allows scalability and security benefits for handling a large number of translation requests and other languages.

AWS AppSync, a fully managed service for developing GraphQL APIs will be used in order to create a GraphQL API that interfaces with translation services. In fact, AWS AppSync can be configured to connect to translation services, such as Amazon Translate, using AWS Lambda for executing the translation logic.

7.2 Backend Services

AWS Lambda, a serverless computing service, is the gateway between AWS AppSync and the translation engines. One of Lambda's capabilities is to handle quick computation and incorporate business logic between the gateway and translation engines.

Finally, **AWS Step Functions**, a serverless workflow service, is as a tool used to orchestrate the translation workflow and managing the flow of data between translation services. Using Step Functions has numerous benefits, including defining dependencies and the ease of managing the application flow using a JSON document.

The architecture of the application starts with **Amazon DynamoDB**, a serverless and highly scalable NoSQL database, which is chosen as the database for storing the URLs of translated blog posts.

Next, **Amazon S3** (Simple Storage Service) is introduced as the storage solution for the actual content of translated blog posts. DynamoDB and S3 are integrated to enable efficient storage and retrieval of translated AWS blog posts.

7.3 Translation and ML Services

Amazon Translate, a natural language translation service, is discussed as one of the translation modes in the project. The Amazon Translate API is used for translating HTML documents while preserving images and other blog-specific elements. The customization feature of Amazon Translate is also utilised in order to keep technical terms untranslated.

Following is **Amazon SageMaker**, a platform for machine learning operations, which provides tools for training, testing, deploying, and managing machine learning models. SageMaker is used in data preparation, preprocessing, training, hyperparameter tuning, model evaluation, deployment, inference, monitoring, and iteration processes in order to build a fine-tuned translation model.

8 Implementation Process

Process text goes here...

8.1 Meetings with the Client

8.1.1 Requirement elicitation phase

8.1.2 Setup phase

8.1.3 Implementation phase

8.2 Data collection and preprocessing

8.3 Integrating Amazon Translate

8.4 Building the Machine Learning model

9 Development Issues

9.1 Version Control

9.2 Temporary Account Access

9.3 Access to Blog Post Database

10 Testing

Process text goes here...

10.1 Testing Frameworks

10.2 Manual Testing

10.3 Unit Testing

10.4 Snapshot Testing

10.5 Integration Testing

10.6 Acceptance Testing

11 Evaluation Solution

11.1 Translation Quality

11.2 Comparison of the two approaches

12 Ethics and Values

12.1 Ethical Approach

12.2 Societal impact of our application

12.3 Possible outcomes resulting from our application

Our full-stack web application uses two different types of translation services: Amazon Translate and our own transformer-based translation model trained on a data set consisting of manually translated Amazon Web Service blog posts. Due to our application relying heavily on machine learning models in order to translate blog posts from English to Turkish, there is room for ethical concern when working with these models.

The translations completed by our model are important to increase the accessibility of information to native Turkish speakers. Through the use of this service, these users should be able to access the same information as any other user that is proficient in a high-resource language. The quality of the translation that our model outputs has a high impact on the user's ability to understand the blogposts; a high-quality translation service will increase user satisfaction and experience by allowing more users access to significant technical information residing in the blog posts. In the case that we achieve a good translation model, more users will be reached through AWS markets which will have a mutually beneficial outcome for our client as well as AWS customers/users. However, in the case that we achieve a low-quality translation, these users will not be given access to important information that they should have.

Although we will be training our model on a manually translated data set, it is significant to consider the difficulties regarding translating English to Turkish since we may not be able to catch the accurate meaning of the sentences with the limited data set we have. Inaccurate translations have the potential to be harmful to users who wish to acquaint themselves with information regarding AWS in terms of time and cost (for example, our model could inaccurately translate the way an AWS service works, leading to time and possible money lost as the user who is reading the blog could carry this mistake over into their work).

In general, we have found that our translation model does not pose much of a concern regarding human rights and values as our model is intended to only work with AWS blog posts that mostly remain technical and related to AWS Services, so there is little room for dangerous translations. Furthermore, since our application is very connected to user experience, before our finished product, we will do acceptance testing to ensure the quality of our translation as well as a good user experience. We will try to improve our translation by gathering more training data from other resources to strive for a high-quality translation service.

13 Conclusion

14 References

[1] "Atlas of the world's languages in danger," Paris, France, 2010, available online: https://unesdoc.unesco.org/ark:/48223/pf0000187026 (Accessed on May 25, 2023).

[2] "Ethnologue: Languages of the world," Dallas, Texas, 2021, available online: http://www.ethnologue.com (Accessed on May 25, 2023).

[3] "Aws blog post count." Message posted to https://chat.openai.com/chat, April 25 2023, asked GPT4 how many AWS blogpost there were prior to september 2021.

[4] "Applied nlp in the enterprise for linguistic diversity: Low-resource language translation using the aws ai/ml stack," 2023, project Description on Project Forum.

[5] P. Koehn, "Enabling monolingual translators: Post-editing vs. options," in *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*, 2010, pp. 537–545.

[6] D. Khurana, A. Koli, and K. e. a. Khatter, "Natural language processing: state of the art, current trends and challenges," *Multimedia Tools and Applications*, vol. 82, no. 13, pp. 3713–3744, 2023.

[7] S. Ranathunga, E. A. Lee, M. P. Skenduli, R. Shekhar, M. Alam, and R. Kaur, "Neural machine translation for low-resource languages: A survey," *CoRR*, vol. abs/2106.15115, 2021. [Online]. Available: https://arxiv.org/abs/2106.15115

[8] M. Alsan, "The best machine translation software you can try in 2023," April 21 2023. [Online]. Available: https://weglot.com/blog/machine-translation-software/

[9] A. Bapna, I. Caswell, J. Kreutzer, O. Firat, D. van Esch, A. Siddhant, M. Niu, P. Baljekar, X. Garcia, W. Macherey, T. Breiner, V. Axelrod, J. Riesa, Y. Cao, M. X. Chen, K. Macherey, M. Krikun, P. Wang, A. Gutkin, A. Shah, Y. Huang, Z. Chen, Y. Wu, and M. Hughes, "Building machine translation systems for the next thousand languages," 2022.

[10] G. Leopold. (2015) Aws rates highest on cloud reliability. Accessed: May 31, 2023. [Online]. Available: https://www.enterpriseai.news/2015/01/06/aws-rates-highest-cloud-reliability/

[11] A. Hudaib, R. Masadeh, M. Qasem, and A. Alzaqebah, "Requirements prioritization techniques comparison," *Modern Applied Science*, vol. 12, 01 2018.

[12] N. Ramadhanti, C. Budiyanto, and R. Yuana, "The use of heuristic evaluation on ui/ux design: A review to anticipate web app's usability," vol. 2540, 01 2023, p. 080008.

[13] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213–1221, 2012, special Issue: Agile Development. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121212000532

[14] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[15] "Software engineering: Sdlc v-model," Feb 2023. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/

[16] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, vol. 50, no. 9, pp. 833–859, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584908000256

[17] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001. [Online]. Available: http://www.agilemanifesto.org/

[18] K. Forsberg and H. Mooz, "The relationship of system engineering to the project cycle," *Engineering Management Journal*, vol. 4, pp. 36–43, 04 2015.

[19] S. Ahmed, W. Channa, and U. Kashif, "Comparative analysis of software process models in software development," 06 2021.

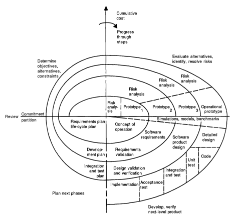
A Research on Methodologies

A.1 Agile

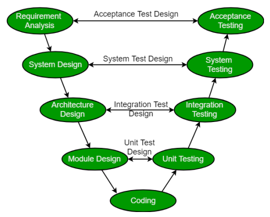
As software engineers from TU Delft, we've been taught to use Agile methods, specifically, the Scrum method, wherein engineers create a feedback loop based on what has been done, and what needs to be done, having regular meetings where progress is reviewed, and brief planning is done for the near future. In most Agile methods, immediate productivity and risk reduction are put to the forefront, which, specifically in the Scrum method takes, place through constant meetings with the client and with the teams, to assess difficulties, changes in requirements, and specifications, all in the name of reducing risks with regards to the final product [16]. Following the release of the Manifesto for Agile Software Development, these methods have become highly popular, as they stray away from the previously bloated, long-winded development models, such as the Waterfall [13]. Agile methods are practical, as they focus on simplicity, constantly working software, and welcome changes in the specifications or requirements [17]. This is where Agile methods strive: simply producing results in environments where the engineers just have to make a working system, however, we find that there are some limitations to Agile and its simple, incremental style. The lightweight framework, although very efficient for simply writing code, has little in the way of documentation, prototyping, and formal studies, making the steps following production (operating, maintenance, and deployment) potentially more difficult than they have to be. As our client is an extremely large company, and they've explicitly mentioned that they hope to continue maintaining, modifying, and deploying our system after we have finished, we find the steps after **our** production life-cycle to be extremely important. For this reason, among others, we explored alternatives to Agile [18].

A.2 Waterfall

The Waterfall model is a process method that, recently, has been less popular due to the rise of practicality-focused Agile methods, however, its apparent decline in popularity does not mean it is necessarily inferior. The Waterfall model belongs to a class of process models called Document-driven models, which, contrary to Agile models, have clearly defined stages, which are demarcated by detailed documents [14]. Although some may not see the value in spending time creating documents for each stage, having documentation to reference later on, whether it be for the same team, new members, or different team picking up where another left off, documentation, especially depicting choices in design are an invaluable resource, which is often not given explicit time in Agile methods. Furthermore, the Waterfall method is linear, assuming that each steps occur one after the other and that teams often shouldn't have to go back, unless something goes wrong. Given that we intend to make a robust system to hand off to AWS after the completion of this project, detailed documents after each are extremely useful. This methodology also has extremely clear milestones, which, although there will not be a working system at each step, the client will have some sort of idea of how the project is going with the documents. Despite this, the Waterfall model certainly isn't perfect. Given the robust and sequential nature of the model, it is rather inflexible, as changes to the requirements, according to the model, are not able to be accounted for without going back and editing every after the first one that was changed. Furthermore, since prototyping and experimentation don't occur until later, errors may be detected rather late, which, given the short-lived nature of this project, we certainly cannot afford [19].



(a) Spiral model [14]



(b) Vee model [15]

Figure 4: The Spiral and Vee models side by side

A.3 Waterfall Derivatives: Spiral and Vee

From the Waterfall model, came many improvements, the two most notable that we have considered are the Vee model and the Spiral model [14] [15]. The spiral model builds on top of the Waterfall by creating an iterative approach in the planning phase, wherein each phase risks are assessed and analyzed, a prototype is created, and more information is contextualized for the next phase, making the successive iterations more detailed, yet time-consuming. This model does not appear to have much in the way of dictating how the implementation steps are to be carried out, as by the time actual implementation is to occur, the documents depicting the software should have been already written, with only the fine-grained details being necessary for the final spiral. Alternatively, for our purpose, the prototypes could be adapted for high-level iterations toward the final working product. The Vee Model builds on top of the Waterfall model by being linear, however, adds explicit many explicit verification and validation steps. The model suggests that software engineering is at the highest level, verifying the requirements at each step, and then after code completion, through testing, work its way back up, going through steps of unit testing, integration testing, etc. [15]. These validation steps work in tandem with the verification steps in terms of abstraction, ensuring that they meet and align with each other, oftentimes being done in parallel. With all this information in mind, it is important to remember the purpose of a process model: "Thus, a process model addresses the following software project questions: (1) What shall we do next? (2) How long shall we continue to do it. [14]"