


2018

A software architecture for cloud-based text annotation: The AFLEX Tag Tool Architecture (ATTA)

Mahmood Ramezani
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/etd>

 Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Ramezani, Mahmood, "A software architecture for cloud-based text annotation: The AFLEX Tag Tool Architecture (ATTA)" (2018).
Graduate Theses and Dissertations. 16863.
<https://lib.dr.iastate.edu/etd/16863>

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

**A software architecture for cloud-based text annotation:
The AFLEX Tag Tool Architecture (ATTA)**

by

Mahmood Ramezani

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Human Computer Interaction

Program of Study Committee:
Stephen Gilbert, Major Professor
Annette O'Connor
Elena Cotos

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2018

Copyright © Mahmood Ramezani, 2018. All rights reserved.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	iv
LIST OF TABLES	vi
NOMENCLATURE	vii
ACKNOWLEDGMENTS	viii
ABSTRACT	ix
CHAPTER 1. INTRODUCTION	1
Purpose of work	1
Motivation	1
Systematic Reviews & The AFLEX Project	3
Software layers of ATTA	5
Evaluation Criteria	6
Multi-User Support	7
Conflict Resolution	7
User-Centric UI Design	8
Input Filetype Support	8
Data Storage Architecture	8
Output Filetype Support	9
Extensibility	9
Thesis Organization	10
CHAPTER 2. BACKGROUND	11
Introduction	11
AI in Publishing	12
AFLEX	13
Machine Learning in AFLEX	14
AFLEX Tag Tool Architecture (ATTA)	16
CHAPTER 3. METHODS	19
ATTA Front-End Development	19
Planning	19
User Needs Discovery	20
Design	21
UML modeling	22
Implementation	24
Documentation	24
Testing	25
Maintenance	26
ATTA Development Summary	26

CHAPTER 4. ATTA TOOLS DEVELOPMENT	28
ATTA Tools Planning and Design	28
AFLEX interviews	28
Stakeholders of AFLEX	29
Expert reviewers	29
Linguists	31
System admins	33
Machine learning researchers.....	33
AFLEX Use Case	34
User Stories	35
Wireframes	37
High-Fidelity Prototypes	38
Version Control	38
Development/Production Environment.....	38
ATTA Database.....	39
AFLEX Tag Tool Architecture Development	41
User Management.....	42
Tag Tool 1	43
Tag Tool 2	45
Tag Tool 2 Review Tool.....	47
De-Duplicator	48
Checklist Tool.....	50
File format conversion	53
AFLEX Tag Tool Maintenance.....	54
Documentation.....	54
CHAPTER 5. DISCUSSION	55
Development.....	55
Sustainable Design In ATTA	55
User interface design considerations.....	56
Challenges	56
The .pdf File Type	56
Evaluation	57
Future Work.....	58
Machine Learning Implementation	58
Data exchange pipeline	58
AI-Assisted Checklist Tool	59
UX Research on Tool Usability	60
REFERENCES	61

LIST OF FIGURES

	Page
Figure 1-1 AFLEX Tag Tool Architecture usage demonstration. Users review and tag excerpts of multiple documents. Their results are stored and passed to others or serve as training data for machine learning algorithms which, in the future, will automatically review and tag documents.	1
Figure 1-2 Layers of the AFLEX Tag Tool Architecture (ATTA). Data Access Layer (DAC) is the farthest from the user and Presentation Layer is the layer that user interacts with. Business Layer connects the other two layers and all the program logic is in this layer	6
Figure 2-1 Steps for creating a systematic review process (Tsafnat et al., 2014).....	15
Figure 3-1 Spiral SDLC (Software Development Life Cycle) model that is used in ATTA	20
Figure 3-2 A sample use case diagram	23
Figure 4-1 The use case diagram of AFLEX tag tools	35
Figure 4-2 Pictures illustrating whiteboarding techniques during the interview sessions.	36
Figure 4-3 A wireframe of ATTA that was used to improve the efficiency of UI design.....	37
Figure 4-4 The ATTA database schema	40
Figure 4-5 AFLEX authentication system using Google services.....	43
Figure 4-6 Screenshot of Tag Tool 1 designed and developed for expert reviewers.	44
Figure 4-7 Comment box and work history module for Tag Tool 1 (left), and work history deletion confirmation message (right).....	45
Figure 4-8 Tag Tool 2, which is designed for linguists.	47
Figure 4-9 A screenshot of the Review Tool, which enables linguists to verify data.	48
Figure 4-10 A diagram of the de-duplicator module in relation to ATTA	49
Figure 4-11 A screenshot of the Checklist Tool	52

Figure 4-12 A screen shot of Checklist Tool output, a PDF file.	53
Figure 5-1 A diagram of the communication between an ATTA tool and an AFLEX ML module.	59

LIST OF TABLES

	Page
Table 4-1 User requirements for expert reviewers (Ramezani et al., 2017).....	30
Table 4-2 User requirements of linguists (Ramezani et al., 2017)	32
Table 4-3 System admin requirement for ATTA.....	33
Table 4-4 Machine learning researchers' requirements for ATTA	34
Table 5-1 A comparison between AFLEX tag tools and similar tools in the literature. N/A indicates that not enough information was available for evaluation.	57

NOMENCLATURE

AFLEX	Automated Functional Language Extraction
ATTA	AFLEX Tag Tool Architecture
GUI	Graphical User Interface
SR	Systematic Review
UI	User Interface
UX	User Experience

ACKNOWLEDGMENTS

I would like to thank my committee chair and my advisor, Dr. Stephen Gilbert, and my committee members, Dr. Annette O’Conner, and Dr. Elena Cotos, for their guidance and support throughout the course of this research.

In addition, I would also like to thank my spouse who has always been supportive, Vijay Kalivarapu for his support and hard work on AFLEX, Glen Galvin who helped a lot with server configuration and security concerns, and the rest of AFLEX team, including Nirav Kamdar, Jingyu Liu. Also, I thank Elizabeth Lee, whose constructive feedback helped with this project. I would also like to thank the Center for Communication Excellence (CCE) at Iowa State University, whose consultants helped me with writing this document. Finally, I would like to thank the HCI faculty and staff for making my time at Iowa State University a wonderful experience.

ABSTRACT

Text annotation is a valuable method of adding metadata to an existing text or document. However, there is no standard text annotation tool across disciplines, in part because of the variety of disciplinary needs. This document presents the AFLEX Tag Tool Architecture (ATTA), a modular software system to allow the development of text annotation tools across disciplines that vary in user interface according to the needs of the disciplinary users, but share a common technical back end, ATTA.

This research describes the development of ATTA, along with the development of four different ATTA-based software tools related to text annotation that meet the needs of different stakeholders: Tag Tool 1, Tag Tool 2, the Review Tool, and the Checklist Tool. All tools are web-based applications that store data to an online database. ATTA-based tools have been found to be useful not only for performing text annotation as its own end goal, but also as a method of data collection for training machine learning classifiers that perform automated text analysis.

CHAPTER 1. INTRODUCTION

Purpose of work

The primary goal of this work is to develop a modular software architecture for text annotation that can be used across disciplines with maximum compatibility and minimal software requirements. While people may be familiar with the ability to add comments to Microsoft Word documents or PDF files, forms of text annotation, these tools do not meet the above goals. This document describes in detail the motivation for this goal, steps taken to meet it, and the final resulting software architecture and tools, which offer an efficient way of collecting data for automated systematic review. The tools are also responsible for exporting data in the requested format with specific desired structure. This architecture was developed as part of a project called AFLEX (Automatic Functional Language Extraction), which is explained in more detail in Chapter 2. Thus, this paper will refer to the AFLEX Tag Tool Architecture (ATTA). The usage of ATTA is depicted in Figure 1-1.



Figure 1-1 AFLEX Tag Tool Architecture usage demonstration. Users review and tag excerpts of multiple documents. Their results are stored and passed to others or serve as training data for machine learning algorithms which, in the future, will automatically review and tag documents.

Motivation

To conduct research on a subject, it is critical to know what has already been done on that topic by other people. Conducting new research risks wasting expense and time if it does

not use results from previous research. However, it can be difficult to keep track of the previous research on a topic. Considering the large number of research publications per year, approximately 400,000 in 2015 in Elsevier journals alone (Reller, 2016), significant effort is required to find relevant information about a specific subject and to manage it.

Since the amount of the information is a lot, one solution to manage the information is to summarize the reports and make note of the most important parts of the different texts. This approach has the potential to make knowledge sharing easier and information management more efficient. This technique, which is known as text annotation, has existed for a quite long time.

Historically, medieval authors of manuscripts used margins of manuscripts or the interlinear spaces as a forum to debate parts of the text, or to share knowledge (Wolfe, 2002). This practice was a critical part of medieval reading so that annotations were usually transcribed along with the original text.

For the purpose of this research, the term “tagging” is used as a subcategory of annotation. Tagging is a method of preparing unstructured text and documents for data analysis and pattern recognition. Text tagging usually involves associating predefined machine-readable labels known as tags, with a specific part of the original text. A row of figures from company's income statement might be tagged as “assets,” for example. A string of text transcribed into a medical chart might be tagged as “diagnosis.” The tags can be used for indexing texts, categorizing them, or adding metadata to manuscripts. The important point is that these tags are usually discipline-specific. For example, the tags that are used by medical researchers are likely quite different from linguists, since tags are typically related to

the specific topics of interest within a discipline. A modular tool for text annotation should be able to support tagging from multiple disciplines.

Not only are the tags themselves different across disciplines, but also the way the annotation is done can be quite different. For instance, some expert reviewers might need to select a single excerpt of the text and assign a tag to that part of the text. This is a “one to one” mapping, e.g., selecting the title of an article and tagging it “title.” In other contexts, e.g., linguistics, people may want to choose a tag for a language pattern that recurs and select as many instances as they find in the text that matches that specific pattern. This is a “one to many” mapping, e.g., the tag “prepositional phrase” may apply to many excerpts. There may also be “many to many” mappings, in which multiple text excerpts combine to serve as evidence of several attributes of the paper. A modular text annotation tool should be able to support multiple text annotation behaviors of users.

Systematic Reviews & The AFLEX Project

Systematic reviews are a research strategy that has been widely accepted and are used to consolidate results from multiple studies within a specific scope, usually to answer a research question (Gough & Elbourne, 2002). Systematic reviews often involve text annotation (Thomas, McNaught, & Ananiadou, 2011). Using systematic reviews, the process of reviewing a large number of research articles can be done in a structured way. Systematic reviews introduce a transparent and scientific process which is replicable (Tranfield, Denyer, & Smart, 2003). The systematic review process can be done manually or be assisted by automation. The partial automation of systematic reviews is further described in Chapter 2.

The current research arises from efforts on a project called Automated Functional Language Extraction (AFLEX), a research effort focused on getting artificial intelligence (AI) further involved into the systematic review process. AI-assisted systematic review tools,

for example, might be able to read thousands of research articles and select only the ones that have specific characteristics that are recognizable by the AI system. To have AI-assisted systematic reviews, the AI algorithm first needs to be trained with a large amount of data. Generally, the more data is fed to the algorithm, the more accurate its decisions are (Banko & Brill, 2001). In the case of systematic reviews, these training data would consist of annotated texts with the desired characteristics manually tagged by experts with the desired characteristics (O'Connor, Tsafnat, Gilbert, Thayer, & Wolfe, 2018). This need provides the motivation for the current research, developing a modular and flexible text annotation system. ATTA allows reliable collection of training data for the AI algorithm.

Based on the literature review in Chapter 2, there is yet another need for ATTA. In some disciplines, the lack of any standardized tool for text annotation, as Microsoft® Word® is for word processing, poses a significant problem. The absence of a standard text annotation tool poses an even greater problem when people from multiple fields need to work together and share data between different applications. This variety in content and usage paradigms poses a non-trivial challenge to design a tool which can be used across multiple fields. Furthermore, when it comes to the software design, there are many details that should be considered. The software should address the needs of its users; however, learning the exact user needs requires tedious effort. The practice of understanding user needs is called user requirements discovery and is described further below.

This research presents the AFLEX Tag Tool Architecture, which has enabled the creation of four different tools to be used across multiple stakeholders with the ability of data exchange among them (Figure 1-2). This architecture consists of three different layers which can interact with each other through application data exchange pipelines.

Software layers of ATTA

The lowest layer, and furthest from user interface (UI), is the data access layer. This layer is responsible for communicating with the database and provide services to upper layers. All the data exchange between the application and database occurs in this layer. Database drivers, service providers, and connection managers are in this layer. In ATTA, the PHP Data Objects (PDO) extension was used to access the database, which supports Object Oriented Programming (OOP) paradigm and adds a great level of security to the application.

The second layer is the business layer in which all the application logic occurs. The business layer is responsible for running the actual code of the application, communicating with the data access layer to exchange the data, and to provide services to the presentation layer or UI. ATTA uses PHP Hypertext Preprocessor (PHP) as the scripting language to function the application.

Finally, the presentation layer of the application, which is the closest to the users, lives at the highest level of the architecture. The users interact with the application using this layer. The presentation layer also communicates with the business layer, providing functionality for the users of the application. There are four different user interfaces designed and implemented using ATTA: Tag Tool 1, Tag Tool 2, Review Tool, and the Checklist Tool. Each tool has a different purpose, with different users in mind, though each is related to the underlying goal of text annotation.

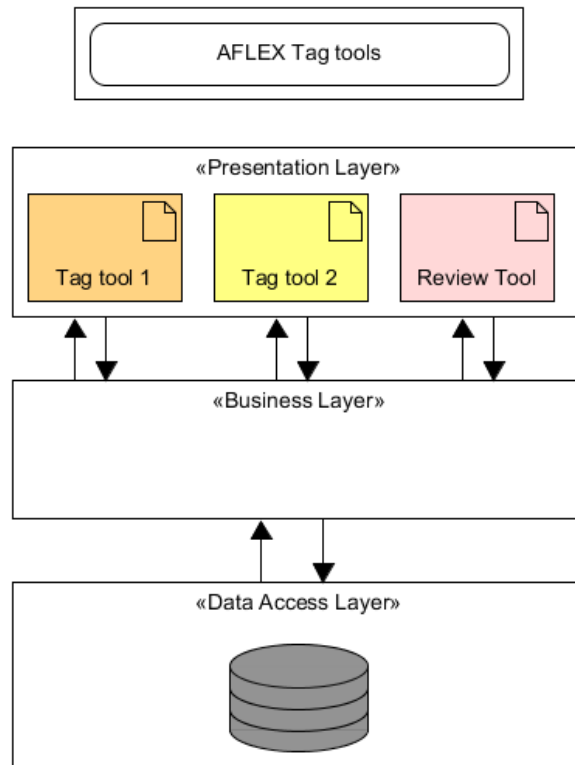


Figure 1-2 Layers of the AFLEX Tag Tool Architecture (ATTA). Data Access Layer (DAC) is the farthest from the user and Presentation Layer is the layer that user interacts with. Business Layer connects the other two layers and all the program logic is in this layer

Evaluation Criteria

During the development of ATTA, the author created specific criteria that could be used to evaluate the resulting tools. The criteria included the following design issues, and each is described in more detail below. While these criteria are not innovative within themselves, this specific combination of criteria is important to note for ATTA. The author compiled this list based on user needs, feedback, and interviews that he did during about two years of development of this project.

- Multi-user support
- Conflict resolution

- User-centric UI design
- Input filetype support
- Data storage architecture
- Output filetype support
- Extensibility

Multi-User Support

One of the most important ATTA user needs is the ability to manage each individual users' work in the application. The task of annotation is often accomplished by multiple people. If the application cannot keep track of the users and what they have done, one would lose the chance of removing biases and normalizing the results, e.g., by comparing and integrating data from different people's work. Also, including this feature can have benefits for the users as well, such as session management and ability to continue their work on an off-and-on basis over a longer time period.

Sometimes an application is designed for one purpose only and not to be used by multiple people. The problem in the case of this study is very different. With text annotation, users believe that they need the same application for different types of text annotation, but they actually need a different interaction mechanism for each purpose. This variety of user needs makes the design and development of the application more challenging.

Conflict Resolution

As it is expected in a multi-user system, there may be cases that different users (reviewers) that worked on the same subject would disagree on a subject. In these cases, there must be a mechanism to identify the conflicts and be able to come into a final decision. For the purpose of ATTA, this function was extremely important as without it, the multi-user

support wouldn't be functional at all. The dataset that results from ATTA should be clean and reliable, as the machine learning algorithms will be trained based on it.

User-Centric UI Design

Reviewing is a tedious task that needs hours of continuous monotonous work on different texts. Thus, it could be exhausting if the user needs to deal with a UI that is not designed based on her needs, as well as the burden of the reviewing task itself. An inefficient UI design reduces not only the efficiency, but also leads to more human errors as the user gets more fatigued (Matthews, Davies, Stammers, & Westerman, 2000). On the other hand, a UI that is designed based on the user needs can be greatly helpful. The learning curve for using the latter UI is shorter, and the user feels less cognitive load working with it. As a result, the efficiency increases, and human errors is reduced. ATTA UI design was done based on its users' needs that were discovered by close communications between the author and application users.

Input Filetype Support

In the domain of manuscripts and digital writing, there are different files types, e.g., .docx, .tex., .txt., rtf., .pdf, and more. Each of them has its own purposes and structure, and its specific way of being handled. Designing an application that would deal with text in general demands supporting multiple filetypes, at least the most common ones. However, the list of filetypes requiring support is defined by the problem domain and user requirements. In the case of ATTA, the most common formats for published papers must be covered, which includes .pdf, .doc, .docx, .odt.

Data Storage Architecture

To create annotation data from a set of texts, one person or a team must spend numerous hours reading the texts and creating annotations. So, it is crucial to protect the data

and ensure data integration. The choices of a data storage system can have a great impact on the high-level system design. Using a database management system (DBMS) offers centralized data management, which ensures data integrity and consistency (MacCormick, 2011). It also supports multi-user access to data which is very beneficial for the purpose of this project.

Another important consideration is the ability to share data in common formats between applications. If the data are stored in a way that they are not exportable to other applications, users' interest in that application may be less, because it is not practical for users to further process their work. ATTA uses a database data storage system to provide data consistency and integrity.

Output Filetype Support

Another important requirement is that the application can communicate with other applications. The connection would not happen if the application does not support common file formats and standard protocols. There are many different applications with different purposes and architectures. If they do not follow the standard communication protocols, there is only a remote chance of finding two different applications that would communicate and exchange information. Thus, an application that is designed for the purpose of this project needs to support common file formats for the output, as well as standard communication protocols. ATTA supports common data exchange protocols and file structures such as JSON, XML, and CSV.

Extensibility

In the Software Development Life Cycle (SDLC) (Lunn, 2003), maintenance, the last step in the cycle, plays a critical role. It is important for software to be improved easily. On one hand, adding new features to existing software can be painstaking if the software

architecture is not designed well. On the other hand, troubleshooting could be near impossible if the code is not clear and there are many libraries with no source available for debugging. ATTA incorporated open source libraries besides its modular architecture to make the application extensible.

Thesis Organization

Chapter 2 discusses the AFLEX project and Systematic Reviews (SR), their benefits, and the requirements of the process. Also, some of the tools that are used in SRs, as well as AI-assisted systems and applications for data collection, is described in Chapter 2. Chapter 3 includes software development concepts that have been followed during this research. Chapter 3 also includes discussions about the importance of the theories behind the development of ATTA. In Chapter 4 offers a detailed explanation of the steps that were taken during the development of ATTA. Lastly, Chapter 5 discusses the outcomes of this project and compares it with a few other similar tools. Also, it includes some of the challenges and limits that were posed challenges to ATTA development as well as recommendations for the future work.

CHAPTER 2. BACKGROUND

Introduction

ATTA spans several important disciplines that are critical to understand to appreciate the tools. The tools that have been developed during this project have already been used in different situations, but there are more potential applications of the tools in other majors.

The main purpose of AFLEX is to aid automatic systematic reviews. Systematic reviews, as the name suggests, usually involve a detailed search strategy that is designed with the purpose of reducing bias by identifying, assessing, and integrating relevant studies on a specific topic (Uman, 2011). It is important not to have bias in the reporting because validity of meta-analysis can be threatened with bias. Moreover, having bias in a report can make the results unreliable for decision making (Dwan et al., 2008). Sometimes systematic reviews also involve a meta-analysis component that uses statistical methods to merge the data from different studies into one quantitative estimate (Uman, 2011). Data integration could lead to a much more efficient decision-making system that would not be possible otherwise.

One good example of systematic reviews is in the domain of clinical studies. It is critical to scientifically verify the results of preclinical studies before conducting human clinical trials. The consequences of human experiments based on invalid results could be disastrous (Kaur, Sidhu, & Singh, 2016). Death, neurological damage, or multi-organ failure of human subjects are just some possible consequences. A systematic review can provide evidence based on the entire research field that a given decision is supported by research evidence.

Another example of using systematic review is a review of studies from 1991 to 2013 that used machine learning for software fault prediction (Malhotra, 2015). Papers were

chosen from seven electronic by a target search string and were reviewed using the systematic review process, and the author concluded that machine learning techniques can predict software fault tendency but that its application is limited. There are many other examples of systematic reviews in other fields that make it a valuable process to be considered. There is also examples of moving towards automation of SRs (O'Connor, Totton, et al., 2018).

The current pace of the literature production sometimes seems to be too fast to manually keep pace with the information production. This mass of scientific publications represents a classic data science problem: an overwhelming amount of data that are difficult to collect, hard to interpret efficiently, and harder to summarize (Thomas et al., 2011). One solution, as it has been accepted in many other domains, is to make use of computers' processing power, for example computer-assisted research writing (Cotos, 2016). However, using automation tools in systematic reviews, or other fields, has its own complications and challenges (Adeva, Atxa, Carrillo, & Zengotitabengoa, 2014).

AI in Publishing

Considering the literature, the author found several applications of AI in the systematic review world which are relevant to this project. RobotReviewer (Marshall, Kuiper, & Wallace, 2016) is the name of a user interface that is coupled with an artificial intelligence classifier that uses machine learning to automatically assess bias in clinical trials. RobotReviewer is capable of reading and processing reports in .pdf format and extracting supporting text for the risk of bias judgement.

The results indicate that the AI system is reasonably accurate and only about 7% behind human reviewers. The results are quite impressive; however, the authors believe that the algorithm is not yet ready to completely take over manual risk of bias assessment.

Nevertheless, the AI can significantly reduce the workload of human reviewers in practice. That is, using the justifications that are provided in RobotReviewer output, human reviewers will need to refer to the full report only if the AI judgments are not acceptable (Marshall et al., 2016).

A different group that included the authors of RobotReview offers another example of automatic systematic reviews (Wallace, Kuiper, Sharma, Zhu, & Marshall, 2016). Population/Problem, Intervention, Comparator, and Outcome (the PICO criteria) are typically defined by systematic reviewers' authors. All the reports that match these criteria will be incorporated and the results from them will be synthesized. However, the procedure of PICO elements identification in the full-text reports is a critical yet tedious step in systematic review process. Thus, the authors tried to use a machine learning approach to help with this step. The results show the efficacy of their algorithm using "supervised distant supervision" (Wallace et al., 2016). However, the next steps, which involve the further processing of the output of their system, are yet to be developed.

There are other examples of using AI in systematic reviews for each specific step in the systematic review process, which is shown in Figure 2-1. Project AFLEX was designed to automate some of these systematic reviews' steps, specifically filtering based on experimental research design, which falls in Steps 7 and 9 of the diagram in Figure 2-1.

AFLEX

As it is discussed above, systematic reviews are not produced quick enough to keep pace with the literature production rate. Most often, the production cost, availability of the necessary knowledge, and timeliness are considered as major reasons for the delay (Tsafnat et al., 2014). To automate the systematic review process and overcome the delay, the parts of the process that computers can get involved in needs to be identified.

The designing process of a systematic review involves two parts: one is technical and the other one is creative. It is important to know what part of a systematic review system could be automated. Figure 1-2 shows the systematic review steps that are suggested for automation. However, not all the processes need to follow the same development plan. The automation of some the steps may appear impossible while some of the other ones are already automated. The important point is that the development of similar tools is incremental (Tsafnat et al., 2014).

AFLEX is one of the major efforts towards developing a robust AI-assisted automatic systematic review system. The ultimate goal of AFLEX is to improve the translation of research findings from scientists to society and to enhance communication between scientists. AFLEX aims to dramatically advance the systematic review process by processing scientific publications and automatically identifying and extracting relevant information from them.

Machine Learning in AFLEX

Like other developed systems in SR, AFLEX also has a machine learning (ML) component, which acts as the artificial intelligence in the system and needs to be trained with data. The data collection for the training is not easy. The data must be collected from specific sources, be sanitized according to the model, and be fed into the system in the appropriate format.

To be more specific, the papers for the ML component of AFLEX needed to be segmented, because a single classifier would not work for all the sections. Each section, e.g., Introduction, Methods, Results, needs its own classifier to be trained. While some researchers have pursued automated document segmentation (e.g. Bui, Del Fiol, Hurdle, & Jonnalagadda, 2016; Harmsze, 2000; Kando, 1999), it is not yet broadly reliable.

Furthermore, to identify the important features of an experimental design for SR (one purpose of AFLEX), one first need to define the experimental design features themselves. It is important to know that these features cannot be expressed simply as keywords. Instead, a feature such as “blinded allocation concealment” is an abstract concept which may be indicated in a research text via one or more disparate phrases in multiple sentences.

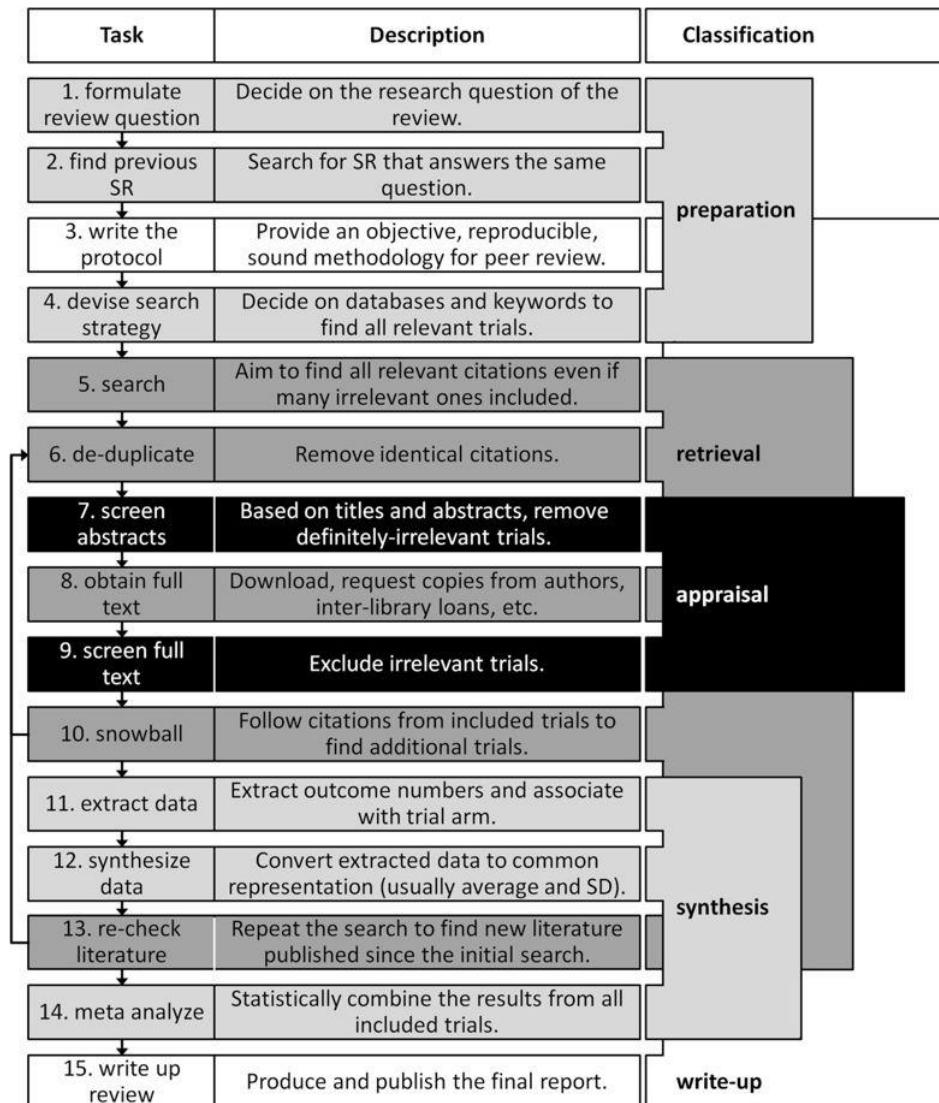


Figure 2-1 Steps for creating a systematic review process (Tsafnat et al., 2014)

After text relevant to the targeted abstract concept is identified, the extracted features need to be further processed according to language patterns. The language patterns are also linguistic features that cannot be identified only by keyword matching. Those features, like “temporal phrase,” require semantic understanding of the text. In project AFLEX, a hybrid of top-down rules based on linguistics and bottom-up statistical machine learning based on n-grams has been developed (O’Connor, Totton, et al., 2018). However, as it mentioned before, machine learning components need to be trained to be able to identify these features from the text. This need is met by the tools developed using ATTA.

AFLEX Tag Tool Architecture (ATTA)

The ATTA project was part of the overall AFLEX project. The goal was to enable expert reviewers and linguists to identify the desired features from the manuscripts and store them in an efficient way, so those data could be used to train the machine learning core. The author first explored the literature to find out if there is already an existing tool for this purpose but did not find any developed tools that would fit the problem criteria. The author mostly looked for the tools for text annotation, as the problem that is described here needs the same interaction and dynamics for text annotation. The challenge, however, was that the interactions needs of expert reviewers were quite different from linguists, two types of stakeholders involved in the AFLEX project.

Several different text annotation tools have been designed for various contexts. The author introduces only a few examples of them to illustrate the current state of the art. BRAT (Stenetorp et al., 2012) is a good example of a modern text tagging tool. This tool is designed to accept plain .txt files as the input and produces the output in .pdf format or as figures. The tool has a clean WYSWYG interface that supports side-by-side comparison of tagged text for conflict resolution between two different taggers. The data structure in BRAT is called

“standoff,” BRAT's term indicating that it does not store the tagged data within the original file. This data structure is suitable for cases that the original file should remain intact.

However, BRAT does not support .pdf files, which is one of the major formats of scientific papers.

Another example of a powerful tagging tool is Callisto Annotation Workbench (Day, McHenry, Kozierok, & Riek, 2004), which is designed for linguists. This tool is open source and written in Java, which supports plug-ins from other languages. Callisto Annotation Workbench allows data comparison and management. The software follows model-view-controller (MVC) methodology that relates the user interface to the underlying data models in an efficient way. However, the input file type is limited to plain text files, and the user interface is cumbersome for users, as it requires multiple clicks from dropdown menus to tag a single text. An example of Callisto Annotation Workbench usage is to tag sentences with rhetorical move/step constructs (Cotos, Huffman, & Link, 2015).

Finally, there is the built-in commenting feature of Adobe Acrobat, which allows storing annotations in the original .pdf files. The shortcoming with this approach is that it is subject to human errors as the user needs to type in the annotation rather than selecting it from a list. Also, as the annotations are stored in the file, the process of analyzing them requires opening each individual file, which makes it a very time-consuming procedure. In the context of the machine learning data collection, Adobe Acrobat built-in commenting feature does not provide a good approach as the licensing prevents accessing the stored annotation data by third-party applications. Another drawback of this procedure is that storing annotation data in the original .pdf files makes it very difficult to collaborate on

annotation; the second reviewer would see what the first reviewer has annotated or, she would need to make a copy of the file and do the annotation.

Considering the examples above and many other applications for text annotation, the author did not find any application that would cover the system requirements for AFLEX data collection. Also, the applications that were discussed above do not support data exchange between different steps of systematic reviews. The ATTA project was defined to enable multi-discipline text annotation, improving upon shortcomings of the other tools and to fill the gap of automation in between some of the systematic reviews' steps. Furthermore, the data structure and communication protocols in ATTA were designed based on the common standards which allows other applications to easily connect and use the ATTA dataset.

CHAPTER 3. METHODS

The creation of the ATTA faced several challenges. In this chapter, the user interfaces that emerged from ATTA are explored as a complex problem in software engineering. The steps taken to come over these challenges were briefly discussed above. In this chapter, the rational and justification for the decisions that were made in different stages of the development are discussed. Lastly, this chapter includes a high-level summary of what has been done during the development of ATTA project.

ATTA Front-End Development

There are various Software Development Life Cycle (SDLC) models in software engineering (Ruparelia, 2010). According to the needs of this specific application, a spiral model was chosen for the development of ATTA. As it is depicted in Figure 3-1, the cycle starts with planning.

Planning

To plan for software, there should be a need. The software is either a response to that need or it could be an improvement or a more efficient way of doing the same task. In the case of ATTA, there was a critical need for an application or a suite of tools that would help different stakeholders to do their desired text annotation tasks. Thus, we needed to define the stakeholders and users. Essentially in the planning phase, we needed to define all the people who were involved or will be involved in the software. Linguists, systematic reviewers (expert reviewers), usability experts, and sysadmins are “actors” in our software ecosystem.

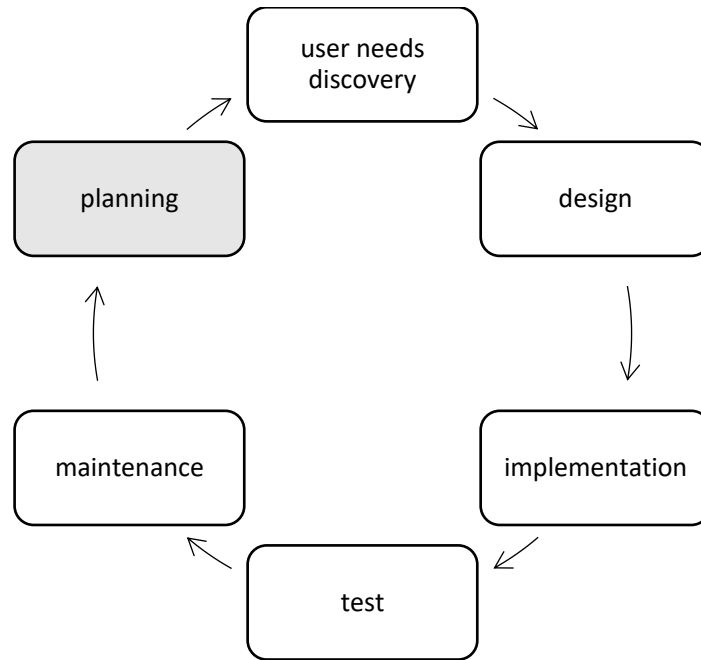


Figure 3-1 Spiral SDLC (Software Development Life Cycle) model that is used in ATTA

User Needs Discovery

After defining important stakeholders in the software planning, the user discovery phase starts. Using user stories in this phase can help to understand user needs. A user story describes the application functions that are needed by a group of stakeholders of the software . User stories can be gathered through user role modeling and user stories are specifically useful when there is no access to all the users of the application.

To understand the user needs, it is also beneficial to interview them about their expectations. Using a prototype of any type during a stakeholder interview can help to understand user needs (Albert & Tullis, 2013).

It is worth to mention that this step is not considered as a common stage in SDLCs. However, the discipline of user-centered design suggests that including this step in any product development process will make the process more efficient and decrease the number of iterations (Baxter, Courage, & Caine, 2015).

Design

After collecting the user needs and expectations, the next step was Design, translating the user needs into the software functions. All the coding would be done based on the design outputs. However, if an application will have a database, one of the first steps is to design the database. A database usually supports all the application data reads and writes and is a critical component of the system. Changes to databases can sometimes be expensive in terms of time. Thus, it was important to make sure that we chose the right type of the database. Currently there are many different types of databases, but at high level, there are two major categories: traditional and NoSQL. There are pros and cons for each, but the type of the database to be chosen for the application is defined by the data type of the application. Usually the data structure and volume define the database type to be used but, NoSQL databases are useful when the data quantity is large and a relational model is not required by the nature of the data (Moniruzzaman & Hossain, 2013). In ATTA, we chose a relational database for our application, which allows secure structured data storage and retrieval.

It is also valuable to note that sometimes the design process cannot be done in one single iteration. Indeed, in most cases, only high-level system designs can be done in one iteration. Then, the software functions will be broken up into different function groups or modules based on the user needs. In each iteration, one of the modules will be taken care of. These partial designs are parts of the system that later will be put together to form the entire software.

Though multiple people contributed to ATTA, it had one primary developer. All the iterations of the tools' development occurred in close communication with users and stakeholders. This active communication reduced the number of iterations and increased development efficiency. Furthermore, following an agile approach in ATTA development

helped to plan and deliver modules in due time, granting stakeholders satisfaction. Agile methods in software engineering offered an answer to the business community that sought quicker software development processes. Agile methods advocates believe that the focus of these methods is simplicity and speed (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003), and typically involve multiple iterations of development based on gathering feedback frequently. In ATTA development, for instance, the development focus was on the functions that were needed immediately to ensure quick delivery. Feedback was collected, and changes were made based on the feedback.

UML modeling

Software design is a collaborative process and there is a common language to be used in this process. UML or Unified Modeling Language (Rumbaugh, Jacobson, & Booch, 2004) is widely used in software engineering. This language has diagrams that help developers to have a consistent way of thinking and design. These diagrams are the views of the UML Model, the model of the software that is built based on the user needs. The practice of putting collected data about the user needs into the form of the diagrams enables developers to model the software more efficiently. For instance, a use case diagram is shown in Figure 3-2. This type of diagram can help developers have a better understanding of translating needs to functions.

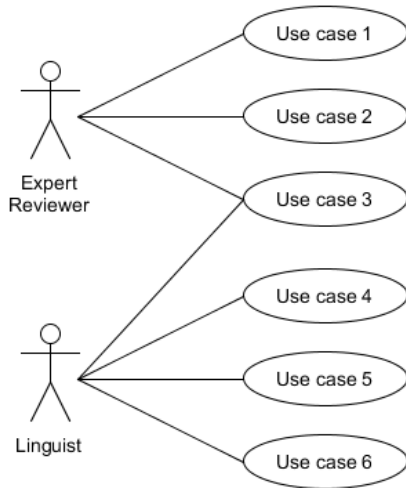


Figure 3-2 A sample use case diagram

In this diagram, there are different actors (users) that interact with the system. Each actor represents a user or a group of the users. The ovals show the ways that users can interact with the system. To give an example related to the ATTA, in this diagram, one actor can represent our expert reviewers and the other can represent linguists. The use cases are the abilities that the users will have in the system. For example, Use Case 3 might be “log in” since both actors share that usage, and Use Case 2 might be “identify experimental design.”

These use cases are all defined based on the user needs and application requirements. It must be pointed out that the use case diagram is not the only type of the diagram that is used in this step. There are other types of UML diagrams such as the activity diagram, class diagram, entity relationship diagrams (ERDs), etc. that can help the design process. In the case of ATTA, use case diagrams were sufficient.

The output of this stage of the SDLC is a model of the system that is translated into different features (use cases) which the final product will have.

Implementation

This step is the main step of development, which means all the coding will be done in this stage. Given the designs from the previous step, each use case will be translated into a software function. Each of these functions are either an application's feature or an answer to a user need. In case of ATTA, the different user types had their own unique ways of interacting with the system. Nonetheless, there are some functions that are shared between different users, such as log in and log out. The specific use cases for the ATTA are listed in Chapter 4.

Documentation

One of the most important, yet most time-consuming practices in software development is documentation. The cost of documentation seems to have been overlooked in the literature (Zhi et al., 2015). In the past few years, before agile methodologies become popular, Rational Unified Process (RUP) (Kruchten, 2004) was one the most common software processes. The documentation requirements in RUP were significant and for each component of the application, there had to be a separate piece of documentation. In contrast, agile methodologies removed this much of focus on the documentation.

Inspired by agile methodologies, the documentation in AFLEX project was done as inline comments in the code. The documentations of this type might be less comprehensive, but in turn, it required less time to accomplished. Also, this documentation is more closely connected to the code, enabling other developers to understand the code faster.

The output of this step was be a working application with limited number of the functions. For the initial iteration, all the functions that should be implemented were in place and they worked as intended, according to the developer's point of view. However, they need to be tested with the end users before they can start using the application.

Testing

There are different types of testing that an application can undergo. There are even some cases in software development in which other applications are responsible for testing the application under development. EvoSuite is an example that does the automatic tests for code which is written in Java (Fraser & Arcuri, 2011).

In case of ATTA, we chose to use a testing method in which every module of the application was tested separately before the delivery and integration with the system. A module is a function, or a set of the functions in the software that can be considered as a unit and has a specific purpose which could be tested against the requirements.

The testing process can involve different activities. It could be a part of the user interface that needs to be tested, or, it can be a part of the back-end functionality. Sometimes there is a condition in which there are modules that depend on each other, but only one of them is ready for testing. For example, a part of the user interface might be ready to be tested, but the corresponding function in the back-end is not ready yet. To make sure that the UI part works correctly, we need to have a response from back-end. In this case, we would create a mock-up function that sends a static response to the front end or user interface. This way, we can test our UI to make sure that it works and, whenever the back-end function is ready, it will be tested and placed in the system.

In ATTA case, the UI was tested multiple times in different iterations before the back-end is implemented into the system. The UI called a function on the server which did nothing but to send a response with a delay to test the UI functionality. The delay in the mock-up response was implemented to simulate connection issues, so that the UI behavior can be assessed in that case.

Maintenance

This step is the final stage to complete a cycle in SDLC. Maintenance involves addressing the issues that are discovered during the testing phase. There also might be some issues not found during the testing, but after delivery, the users would report those issues. In either case, the issues will be resolved in the maintenance phase, and the cycle completes. Another cycle is started as soon as one is ready to add more features to the system or implement another set of functions. All the new functions will go through the same cycle process, starting from planning and, will be in place after the testing is done.

Occasionally, we need to go back and forth between testing and maintenance. For instance, there may be a critical function of the system without which the software would be disrupted. In that case, we need to focus on that function and test it as many times as needed to ensure its functionality. Also, there may be some other critical functions that are important not because the system depends on them entirely, but because of the sensitive information that they handle. In that case as well, the cycle might be limited between test and maintenance steps until that function runs correctly.

ATTA Development Summary

Project ATTA was started in Dec 2016 as a part of the AFLEX project. The development was carried on for almost two years. During this time span, development meetings were held every week with the individuals who oversaw ATTA. In these meetings the progress on the software development was discussed and feedback collected. Also, there were monthly meetings with AFLEX team to make sure ATTA is in sync with AFLEX requirements. Moreover, there were many occasional meetings, roughly one per month on average, with end users of the ATTA tools. Meetings with users usually involved a prototype, a UI design, or a tested module of the software that was ready to be tested by the

end users of application. These meetings helped significantly with speeding the iterations of software development. The total number of stakeholders who were involved in different meetings was 12.

Project ATTA started with the development of the Tag Tool 1. It took almost a semester to complete. Meanwhile, during the meetings about Tag Tool 1, the need for Tag Tool 2 for linguists was discovered. The development of Tag Tool 2 also took almost a semester, which also involved the Review Tool as a needed extension for it. The third semester was spent on development of Checklist Tool, which was not initially planned. It should be noted that during the development of each new tool in ATTA, the previous ones were still under the maintenance iterations for troubleshooting or adding features.

CHAPTER 4. ATTA TOOLS DEVELOPMENT

In this chapter the actual steps that were taken for the application development are explained in detail. This chapter essentially describes the practical implementations of the theoretical discussions of software development in the previous chapter.

ATTA Tools Planning and Design

The first step of planning process is to identify the problem that the application will be designed to solve. The problem of this application was that people needed a dataset of annotated research articles for their research, but there was no efficient way of collecting data. The data needed to be stored in a structured way that could be exported for various purposes.

Considering this short description of the problem, there is a need for a robust data storage mechanism and data-oriented design. Also, there should be a way to enable users to store/retrieve data. The part of the application that allows users to interact with the data or system in general is usually called GUI (Graphical User Interface) or UI in short.

AFLEX interviews

To have a good understanding of the problem domain, the author conducted initial interviews with stakeholders who oversee the project. After defining the project scope, the interviews with users of the application started. As mentioned above, there were regular meetings with stakeholders and the future users of the application. The difference was that the meetings with the users usually involved a prototype for feedback or a deliverable to be tested. Other stakeholders were updated about the project progress and made sure that the project was done within the defined time and cost. Also, they made sure that ATTA met the AFLEX requirements.

Stakeholders of AFLEX

To start designing a UI, all the users of the application must be defined. It should be noted that the users are stakeholders of the application who directly interact with the system through the UI. However, there are other stakeholders who affect the design of the system, though they would not interact directly with the application. For ATTA, the following stakeholders were defined.

Expert reviewers

This group of stakeholders are also users of the application. It means that their expectations and needs will affect the design of the UI. To know the user story of this group, the author interviewed them. A user story follows.

Pat, an expert reviewer, wanted to identify text excerpts of an article that provided evidence that the article authors used a randomized parallel design. Pat found the text “Animals were randomized to treatment groups and induction of transient focal cerebral ischemia,” which Pat tagged as “Random.” Next, Pat found the text “The individual performing the infarct volume analysis was blinded to treatment group,” which Pat tagged as “Blinded Outcome Assessment.” These annotations would be used later by the expert reviewer in a meta-analysis to identify papers in a larger corpus that had randomized designs with an animal population.

A summary of expert reviewers’ requirements is listed in Table 4-1. This table is compiled based on the information that were collected during the interview sessions.

To work on the manuscript, the expert reviewer would need a user interface to store and render the manuscripts. The manuscripts are in (portable document format) .pdf format.

Table 4-1 User requirements for expert reviewers (Ramezani et al., 2017)

#	User Functional Requirements	System Support & Response
1	Read PDF and visually locate text.	PDF rendered at readable size in browser with zooming features
2	Select text within PDF	Selected text highlighted in PDF; plain text extracted to Annotation Box sidebar.
2a	Option: select additional non-contiguous text and they will be grouped together	Additional text highlighted in PDF and plain text extracted and added to Annotation Box.
2b	Option: Delete text selection just made	Text passages in Annotation Box can be deleted individually
3	Select one or more tags	Tags highlight when selected
3a	Option: deselect one or more tags	Selected tags de-highlight when clicked again
3b	Option: add free response comment text	Textbox accepts plain text.
4	Click OK to complete tagging	Text passages and tags stored in database and added to Work History in sidebar. Highlights clear from PDF. Tags reset to unselected. Annotation Box cleared.
5	Note the time spent on tagging	The system records the time taken between OK button clicks to measure the time spent to tag each text excerpt. It is also possible to aggregate the times for a specific document or part of the corpus.

Then, they would want to read the manuscript, find the parts of the text that represent specific design patterns, select that part of the text, and assign the appropriate predefined design patterns (tags) to it. This set of text excerpt and the design patterns (tags) should be stored as an annotation record. They would also need to be able to review the annotation history and make changes if it is necessary. Annotation records also need to be related to the manuscript that they were extracted from.

There are multiple expert reviews that would work on the same set of manuscripts. Therefore, the system needs to keep the track of each individual user's work history and store those data in an organized way. This multi-user dynamic can raise a potential conflict situation. There are two types of potential disagreement between the reviewers:

- 1- Expert reviewers would select the same part of the manuscript that they think represent a specific design pattern but, the design patterns are different from one reviewer to another. (Reviewer A: Text1 > TagA; Reviewer B: Text1 > TagB)
- 2- Expert reviewers would select different parts of the manuscript as the representatives of the same design patterns. Reviewer A: Text1 > TagA; Reviewer B: Text2 > TagA)

Considering these conflicts, there must be a module built in to the system that would enable expert reviewers to resolve the conflicts. Based on the results of conflict resolution, annotation records should be updated to match the final decision. It should be noted that the final decision would likely be made by someone else to make sure there is no bias in the results.

Linguists

This group of stakeholders are also considered as users of the system. Thus, the UI design will be affected according to their needs and expectations. This group's user story is as follows. Also, linguists' requirements are compiled as Table 4-2 based on the information collected from the interview sessions.

Lingu, a linguist, wanted to identify language patterns in the excerpts from the articles that have been annotated by expert reviewers. Lingu was presented with this sentence: "The individual performing the infarct volume analysis was blinded to treatment group." Lingu selected the "Manner" language pattern and marked "blinded to treatment

group” as the evidence of that linguistic pattern. Lingu did not see any repeated records of different expert reviewers extracting that same sentence. Also, Lingu had the context available from which the expert’s text was extracted in case further analysis was needed. Language pattern identification is a part of the text processing to collect data for training machine learning algorithms.

Table 4-2 User requirements of linguists (Ramezani et al., 2017)

#	User Functional Requirements	System Support & Response
1	Read plain text passage.	Plain text rendered at readable size in browser.
2	Select tag to work with.	Tag highlighted in sidebar.
3	Select string of text.	Text highlights when selected
3a	Option: Select an additional string of text that is not selected.	Selected tags de-highlight when clicked again
3b	Option: Delete highlight from a string.	A selected highlight shows a button for deletion.
4	Click OK to complete tagging	Text passages and tag stored in database. Highlights clear from text passage. Tag changes to “tag used” color and gains badge number with count of strings tagged.

The linguists have developed a model based on move/step for methods section (Cotos, Huffman, & Link, 2017). Also, like expert reviewers, linguists have tag sets that are called language patterns. These tags need to be defined before the process starts. The linguists would need to process the annotations even further at the word/phrase level. It means that they would need to choose a language pattern (tag) and then select all the occurrences in the text excerpt according to that specific language pattern. The occurrences should be stored in a structured way, so they can be processed.

Moreover, the results need to be reviewed and changed if necessary. There might be some language patterns that are missed in the text, so they need to be added later through the review user interface.

System admins

Unlike two previous stakeholders, this group of stakeholders are not direct users of the application. Thus, they would affect the UI design. However, they will affect some critical parts of the software. Through an informal interview the author found some technological limits imposed on the project that would limit the development technologies to PHP and jQuery. For example, the available server would not support .Net technologies or NodeJS. In addition, other limitations arose throughout the project. For example, there were some open source modules that needed to be installed on the server, but because of security concerns, the author was not able to do that. The table of system admins' requirements is shown in Table 4-3.

Table 4-3 System admin requirement for ATTA

#	Stakeholder Functional Requirements	System Support & Response
1	Linux on servers	Cannot use Microsoft technologies like ASP.Net, neither NodeJS.
2	Need language and libraries that are familiar.	Limited to use PHP, jQuery
3	MySQL database on the server.	MySQL database should be used for application.
4	Security concerns	They must be taken care of

Machine learning researchers

This group of people form the last group of stakeholders involved in AFLEX project. These users would not need a clean and customized UI to interact with the system, but rather they needed a structured output for their machine learning algorithms. The output needed to

match their desired format with machine readable structure and the data that they required. Thus, there was a need for an output creation (export) module. The output for machine learning is usually in plain text format with the structure defined by the destination application. A tabular summary of machine learning researchers' requirements is shown in Table 4-4.

Table 4-4 Machine learning researchers' requirements for ATTA

#	Stakeholder Functional Requirements	System Support & Response
1	Data format specifications	Data should be exported in the format that is expected by ML researchers
2	Data exchange dynamics	It must be discussed to make sure the data will be exchanged without any problems
3	Data structure specifications	The data needs to be structured to match ML researchers' expectations
4	Security concerns	They must be taken care of

AFLEX Use Case

According to the stakeholders discussed above, use case diagram of the AFLEX will look like Figure 4-1. Note that the only stakeholders that affect the system design will appear in the use case diagram since they are considered as actors in the system.

It should be noted that there is an abstract actor in the use case diagram which is called “user.” The “user” actor performs the use cases that are common between other actors who refer to it. This abstract actor helps organizing the system functions and modules.

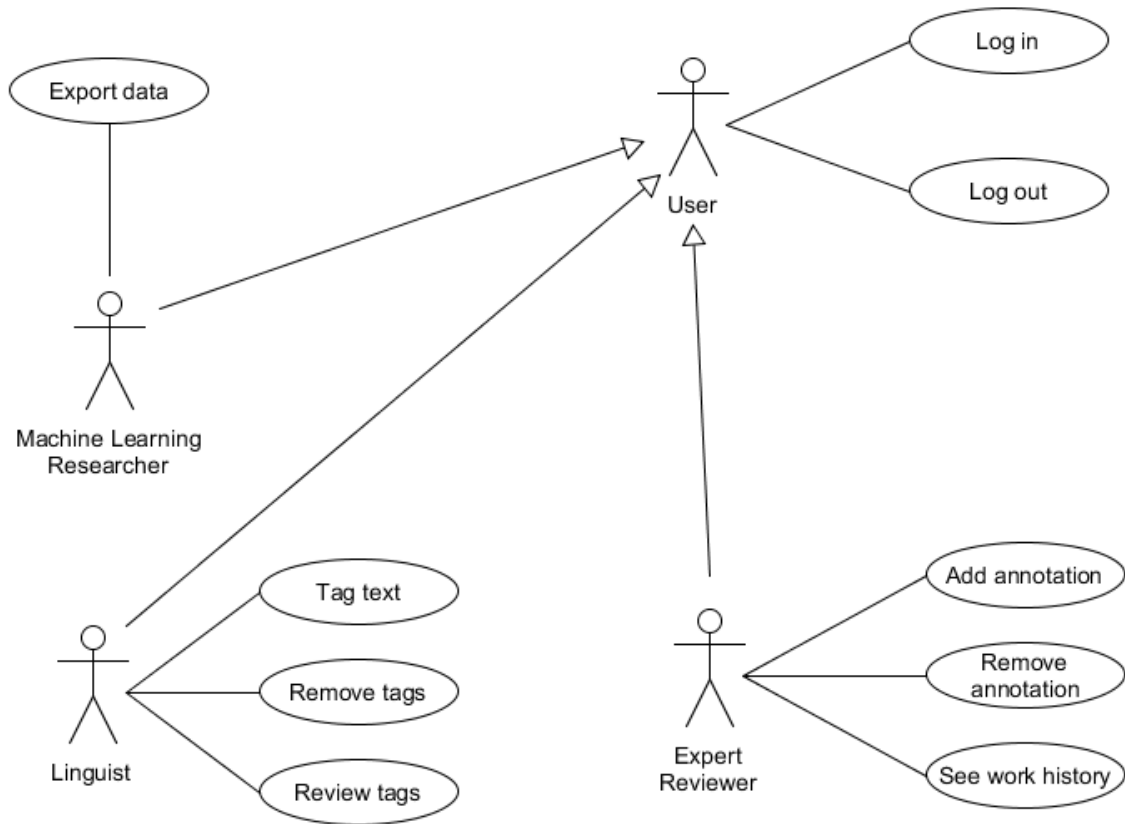


Figure 4-1 The use case diagram of AFLEX tag tools

This diagram helped to design ATTA. After understanding the user needs and interpreting them as action verbs in a use case diagram, it was easier to define the functions and software modules that were needed in the system.

User Stories

Whiteboarding, was used to increase the efficiency of the interviews. Drawing pictures may help people to think in a more organized way by creating "boundary objects" (Carlile, 2002) that allow people with different backgrounds to discuss a situation with common terminology, crossing the boundaries of their disciplines of expertise. In Figure 4-2, there are pictures of whiteboarding during the interviews.

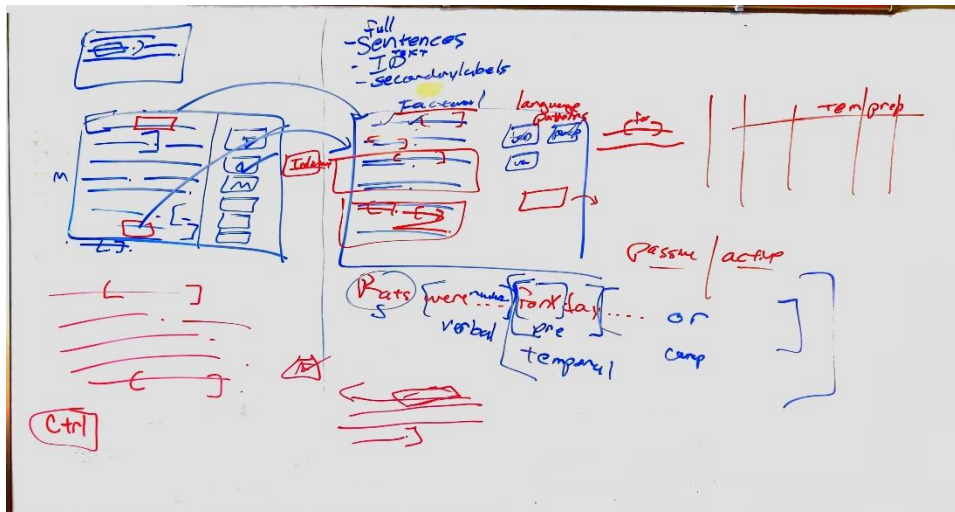
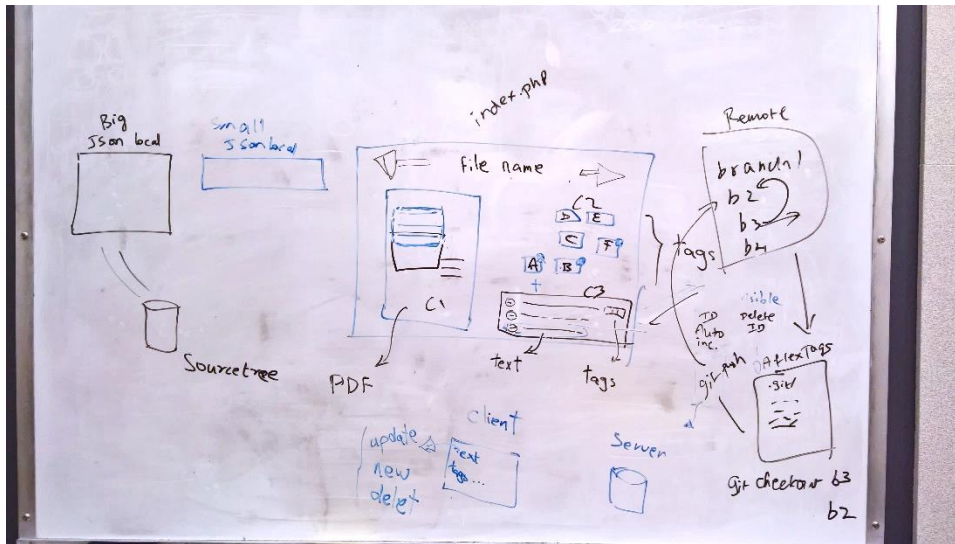


Figure 4-2 Pictures illustrating whiteboarding techniques during the interview sessions.

This step was one of the most important stages of the software development. Using techniques like whiteboarding not only helped communication with the user, but also gave the developer an idea of the users' expectations. The user may expect the software to look like what she draws on the board. Thus, the layout draft of the UI can be defined by the user at very first steps of development.

Wireframes

Using wireframes, the UI designer can create a fast prototype that will give the user the feel of the final product (Albert & Tullis, 2013). In the ATTA project, multiple wireframes were created and discussed with the users. The feedback from the users then used for the development of the final product. One of the wireframes that used during the design process of ATTA project is shown in Figure 4-3.

Using wireframes made the UI design process smoother and more efficient since the communication process with the end user was easier. Also, the interaction with the system after implementation was more intuitive for the users because not only it looked familiar, but also it was more likely to match the mental model that users had in their minds.

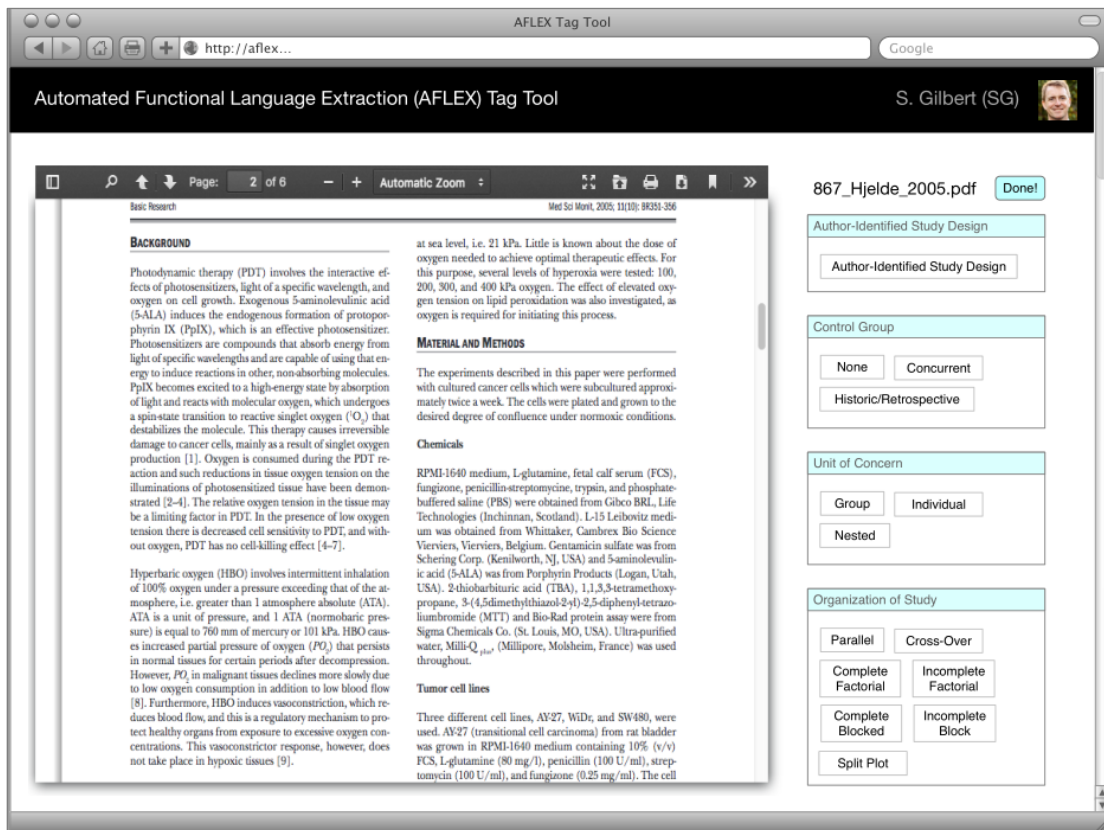


Figure 4-3 A wireframe of ATTA that was used to improve the efficiency of UI design

High-Fidelity Prototypes

Based on the wireframes, the author created high-fidelity prototypes to be tested by the end users. High-fidelity prototypes usually are used with mock data just to show the functionality of the software to the user. The user can interact with the prototype just as she does in the real product, but the data is mocked up.

These prototypes usually had most of the functional elements of the final UI design. The author walked the users through different prototypes that had been developed to make sure that they will match the users' mental models. This step was the last step before designing the final product front-end. It should be noted that this cycle of wireframing, prototyping, and revising (maintenance) happened multiple times through the SDLC of ATTA, as is predicted by agile programming methodologies.

Version Control

The management of changes to the application is called version control. Using version control, developers not only are able to keep track of the changes in the source code, but also are able to revert to a specific version if needed (Loeliger & McCullough, 2012). There are different software solutions for version control. The author chose to use git to manage the changes of the application. One good reason of choosing git is the ability to create branches. By creating branches for the features that are being worked on, the main application remains unchanged. After the new feature or bug fixing is done and tested on a separate branch, that branch can be merged to the main branch and will affect the final software.

Development/Production Environment

Once the system is released to be used, it cannot be down for a long time. The maintenance should occur seamlessly without interrupting the usage of the system. The

development team set up two different environments for the application. These two environments were essentially two identical servers with different purposes.

The first setup was the “development environment.” The goal of this setup is to deploy early releases of the application and to test new features. All the new features and modules were tested through the development environment before being published on the production environment. The user data logged in this environment were useless and were subject to removal at any time. Thus, the users felt free to do whatever they wanted with the system. It is good to note that this environment acts as high-fidelity prototype, and it is usually valuable if system engineers watch the users as they interact with the system at this stage.

The “production environment,” on the other hand, was designed to be used with real data. The data in this environment were securely stored and users knew that they were interacting with the real system. Any software updates to the application took place at low usage times after the updates were tested on the development environment. The production environment was supposed to be up and running all the time and all the common maintenance practices were done for it.

The front-end of the system has been discussed, but there are other critical parts of the system that need to be described. One of the most important layers of the system is the data layer. The data layer is responsible for storing the data and serving other parts of the application with data.

ATTA Database

One of critical decisions for the system designer is to choose the database type. This decision affects all other parts of the system. In the case of ATTA, because data are highly

structured, the author decided to use a relational database. Considering the technologies that were available on the server, the author chose MySQL database for this application.

To design the database, all the entities in the application domain were listed. Usually each entity had a table counterpart in the database. The database design was carefully created, as the application performance highly relies on the database structure. Furthermore, database tables were normalized according to the normal forms. The practice of normalization restructures the database tables to improve data integrity and reduce data redundancy in the database. The results from designing and normalizing the database tables are illustrated in Figure 4-4.

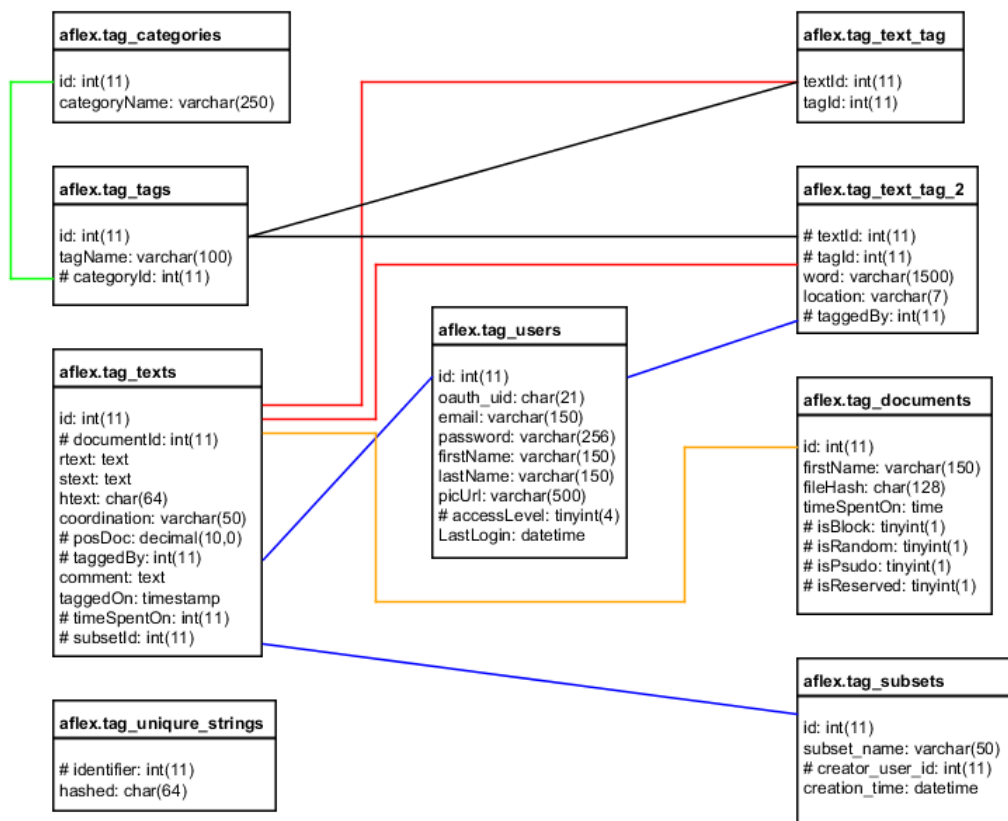


Figure 4-4 The ATTA database schema

As it is shown in Figure 4-4, there are multiple tables in ATTA database. The “tag_” prefix indicates these tables belong to ATTA since there were other tables in the same database as well. Each table is responsible for storing a specific data type in the application. The excerpts from the texts that were annotated by expert reviewers are stored in “tag_texts” table while the document information is stored in “tag_documents.” The “tag_users” table contains information about the users of the application, and “tag_unique_strings,” designed by Philip Cordova, stores the unique strings that are identified by a de-duplicator module. The de-duplicator is explained in detail further below in this chapter.

AFLEX Tag Tool Architecture Development

After the designing phase was done, the development started. In the development phase, all the coding was done based on the results from the designing step. Considering the user needs, the author decided to have the software solution as a web application due to the following reasons.

- 1- There is no need to install anything on the client computers with a web application. Users will need a working Internet connection and a browser to use the application.
- 2- Since the application is at the early stages of development, it is subject to change frequently. Thus, having a web application, which has minimal maintenance costs, is desirable.

According to the technologies available and the limits imposed on the project, the author chose PHP as server-side scripting language. No libraries on the server for PHP were used. Also, JavaScript was chosen for client-side scripting. Pdf.js was chosen as the library for rendering pdf files in a browser, jQuery was chosen for the client-side scripting library, and for the front-end, the Twitter Bootstrap CSS library was used.

User Management

The tag tool needed a user management system. User management systems allow users to sign up, to be authenticated, and authorized (Boyanov, 2005). Furthermore, user identification is a part of the user management that enables the application to track the work history of individual users.

In the case of ATTA, user management is done via Google. Thus, ATTA users need to have a Google account to be able to use the system. Iowa State University is partnered with Google for the university's email system and because of that, most ISU community members have a Google account already set up.

ATTA users should be authenticated in Google servers and authorized through Google OAuth 2.0 service. The Google authentication user interface integrated into ATTA is shown in Figure 4-5. The connection between the application and Google services establishes directly after a user logs in to the system. Google sends the profile information that is requested by ATTA, so user information such as user id, profile picture URL, etc. is stored in the database. It should be noted that the structure of the ATTA database supports independent user management as well.

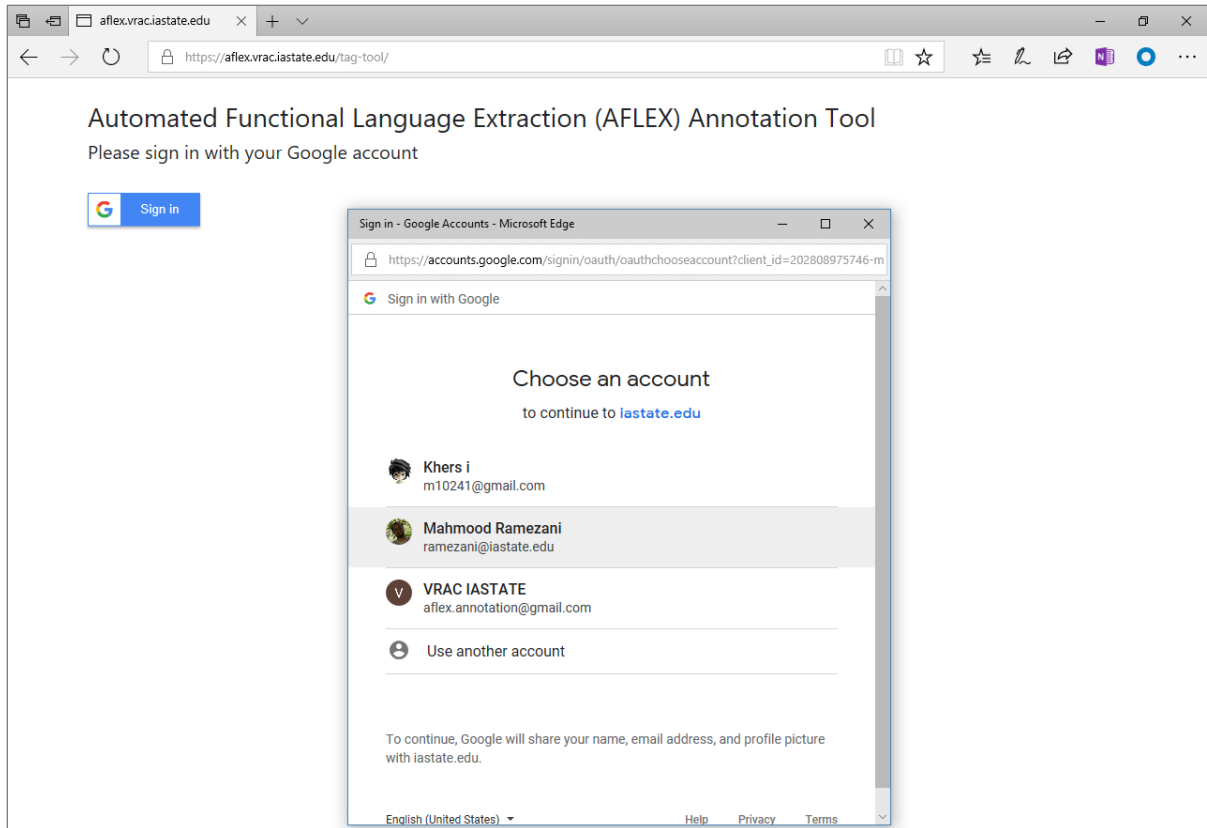


Figure 4-5 AFLEX authentication system using Google services

Tag Tool 1

The first module of ATTA, which is called Tag Tool 1, is an answer to the expert reviewers' needs. They wanted to be able to review a manuscript to identify the design patterns that are present in manuscripts. The parts of the text that serve as evidence of a design pattern should be extracted and stored as an annotation record. The context of the excerpts also is important. Thus, a link to the manuscript also needs to be stored.

Figure 4-6 shows a screenshot of AFLEX Tag Tool 1 in which the manuscript is rendered on the left side panel. On the right panel there is an annotation box which includes that predefined design patterns. These patterns are stored in the database and are loaded

dynamically as the application loads up. Thus, the patterns could be removed/changed easily. Also, it is very easy and fast to add new patterns to the system.

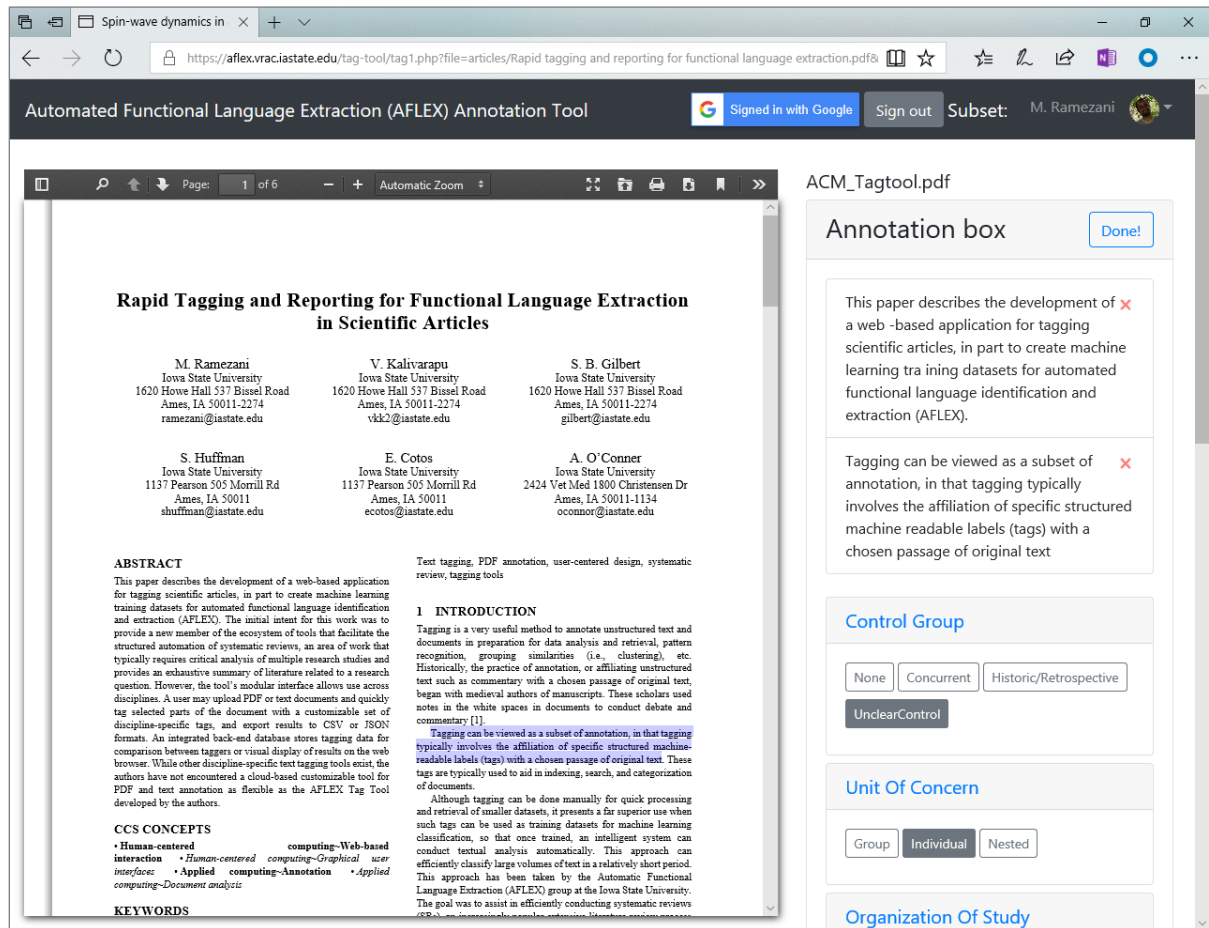


Figure 4-6 Screenshot of Tag Tool 1 designed and developed for expert reviewers.

Furthermore, near the bottom of the right panel there is a comment box that allows users to add comments to the annotation records (Figure 4-7, left). The comment is useful for sharing the rationale for the tag, so the conflict resolution process will be much easier. At the very bottom of the right panel there is work history module UI. This module keeps track of individual reviewer's work (text annotations) and enables reviewers to change or delete their work by clicking on the desired record. The red X button for each record will remove the

record from the database, as would be expected. A delete confirmation message is also included (Figure 4-7, right). It also should be noted that this layout of application was developed based on the wireframes and the feedback from the users of the application.

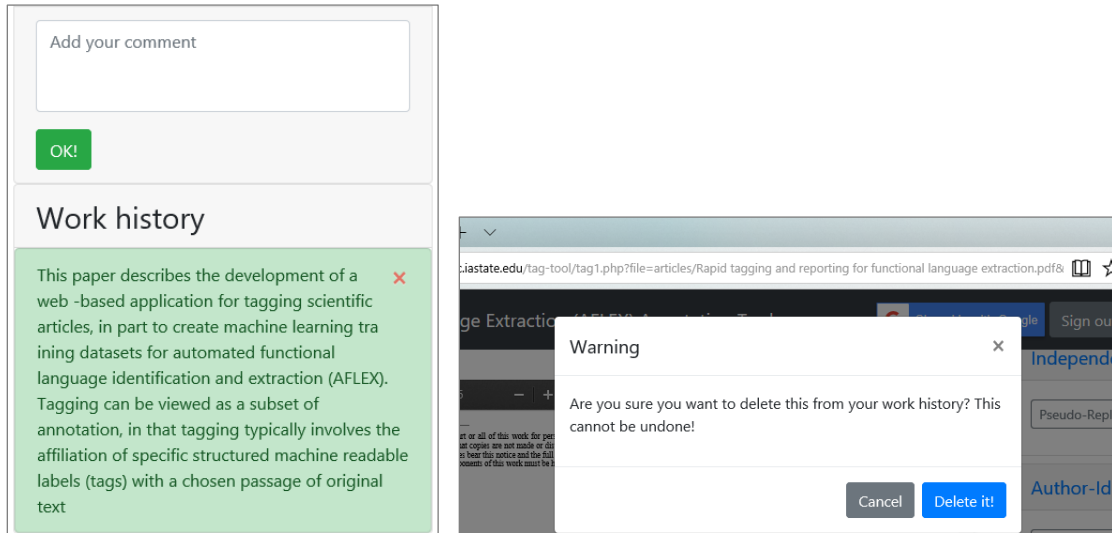


Figure 4-7 Comment box and work history module for Tag Tool 1 (left), and work history deletion confirmation message (right).

Tag Tool 2

Another module of the ATTA is called Tag Tool 2, which is designed for linguists. Essentially this tool will use the output from Tag Tool 1 (annotation records) and will allow linguists to process the records at a word or phrase level. The linguists needed the annotation records along with the text that they have been extracted from. By processing the annotation records, linguists can identify the language patterns in the text excerpts from the Tag Tool 1. This tool's UI is designed similar to Tag Tool 1 to maintain consistency. A screen shot of Tag Tool 2 is shown in Figure 4-8.

Unlike in Tag Tool 1, in which the annotation process starts by selecting an excerpt of the PDF text, in Tag Tool 2 the tagging process starts by selecting a language pattern on the right. The selected language pattern would be highlighted and the user (linguist) selects all

the instances of that particular language pattern in the current annotation record. The user is also able to remove the record by hovering over highlighted part of the text and clicking on the X icon that appears.

The left side is divided into two panels. The top left panel shows the annotation records, one at a time. The annotations that are shown there are the ones that are refined and agreed on by the conflict resolution process. Also, at the bottom of this panel, the total number of annotation records and the current record are shown.

To make the tagging process more efficient and easier for linguists, the annotations were displayed one by one using the Next button at the bottom of the top left panel. The users also can go back to the previously tagged sentences by using the Previous button. Furthermore, the tagging records were seamlessly stored in the database using AJAX, in real time as the user (linguist) selected phrases. This approach was discovered through multiple iterations to be the best tagging interaction process.

The bottom left panel renders the original text from which the excerpt is selected. The reason of rendering the original text is that linguists needed to be aware of the context of the text excerpt. The annotation record is highlighted in the rendered manuscript for convenience.

Like design patterns, language patterns are also dynamically published from the database to the right panel. Thus, they can easily be added, removed, or changed. The only difference from Tag Tool 1 is that in Tag Tool 2, only one of the language patterns can be selected at a time.

Figure 4-8 Tag Tool 2, which is designed for linguists.

Tag Tool 2 Review Tool

As mentioned in previous chapters, the data that is collected needs to be verified to ensure the quality. Sometimes in the verification process, there are some changes to the data that need to be made. Also, there might be some missing points or data points that need to be removed. All these data manipulations need to be done in the same context as the data is collected. Thus, the third user interface is created for this purpose (Figure 4-9).

Figure 4-9 A screenshot of the Review Tool, which enables linguists to verify data.

The Review Tool loads the records from Tag Tool 2 that are stored in the database. The tags are color coded to facilitate the review process. The records are grouped by the file that they are come from, and each excerpt has its records in the same row. A user can modify a specific record by clicking on it. When a record is selected to be modified, it is highlighted in the text to show the user the phrase in context.

De-Duplicator

The De-Duplicator module is one of the underlying functions of the system which is implemented in the business layer of the application. Originally, this module had not been planned. However, after delivery of the Tag Tool 2, a new requirement was discovered. The problem was the repeated text excerpts in the Tag Tool 2 data inputs. As it is mentioned

before, the input of Tag Tool 2 comes from the output of Tag Tool 1, and since there were multiple expert reviewers working on the same dataset, duplicates are expected. However, linguists did not want to see and tag repeated records, as doing so potentially leads to inconsistency, less efficiency, and increasing probability of error.

To resolve the issue, the De-Duplicator module was developed and verified by Philip Cordova, an undergraduate student at Iowa State University. The module uses the SHA-256 hashing algorithm to identify unique records. As one record is added to the Tag Tool 1 table in the database by expert reviewers, the hash is calculated and compared against the rest of the hashes to see if it is a unique record. If the added text excerpt is identified as unique, it is added to the unique sentences table in the database. Otherwise it is skipped by the de-duplicator module. A diagram of the system including the de-duplicator subsystem is depicted in Figure 4-10.

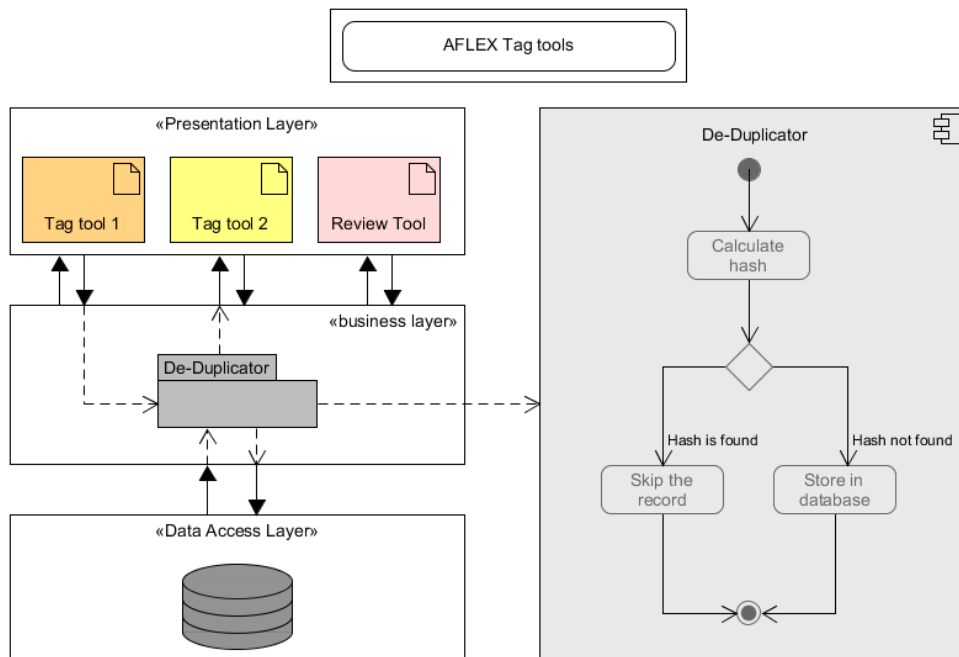


Figure 4-10 A diagram of the de-duplicator module in relation to ATTA

Checklist Tool

One of the most important and powerful features of AFLEX tag tool that is discussed in this document is modularity. Modularity makes ATTA a flexible architecture, allowing it to be adopted and used for various purposes with minimal changes and cost. A good example of this feature is Checklist Tool, which is also developed by the author. The Checklist Tool uses the underlying ATTA structure and the overall goal of text annotation, but with a very different type of tool. The need and justification for creating Checklist Tool is as follows.

In the scientific publication domain, there is a high-impact problem that leads to research wastage. There are several reasons that can cause, for instance, roughly 85% of healthcare research funding to be wasted. Reasons include poor study design, poor research question selection, and poor reporting. In 2010, it was estimated that 200 billion USD was the cost of wastage (Okumura, 2016).

Focusing on reducing the wastage that is due to poor reporting, major medical journals implemented reporting guidelines or checklists which specify a minimum set of items required for a good report. Over 400 reporting guidelines have been developed so far for different types of research (The EQUATOR Network, n.d.). To name but a few, the STROBE (Von Elm et al., 2007) statement is developed for observational studies, while CONSORT statement is created for randomized controlled trials (Schulz, Altman, & Moher, 2010), and the PRISMA statement for systematic reviews (Moher, Liberati, Tetzlaff, & Altman, 2009).

Even though these guidelines have helped reduce wastage, the implementation of a system to use these guidelines has not been convenient. For instance, if someone wants to publish in one of the journals that require completed checklist, she would need to download

the checklist (in .pdf or .docx format), complete it (either by printing it and handwriting or filling up the .docx version), convert it to .pdf format, and finally upload it along with the manuscript. This process is tedious and frustrating and adds to the burden of writing the manuscript itself.

Considering the flaws in the checklist completion process, the AFLEX team designed and implemented an innovative solution that makes the process much more efficient and user friendly. The solution involves adopting ATTA to the problem needs and creating a new tool that helps checklist completion, the Checklist Tool. Using the Checklist Tool, a user uploads her manuscript to the system, completes the checklist using the tool, and downloads the .pdf output that is generated by the system. The output of the Checklist Tool is structured and laid out to be ready to be uploaded to the desired journal.

The Checklist Tool uses a similar web interface as the other AFLEX Tag Tools do. Also, the database and layers of the application are the same as AFLEX tag tools. Despite this, there are significant differences in the user interaction, purpose, and the output of the Checklist Tool and the Tag Tools. A screenshot of the Checklist Tool is shown in Figure 4-11.

One possible user scenario for the checklist tool is when a user wants to upload his manuscript to a journal that needs a completed checklist. After finalizing his report, he goes to the checklist tool, uploads the manuscript and selects the desired guideline from the top left corner. The corresponding items of the selected guideline appear on the right panel. If the user is unsure of what an item requests, he can mouse over the "?" icon to see a tooltip about that item. The user then selects and opens an item from the checklist on the right panel and finds the corresponding text in the manuscript. By selecting the related part of the text, it is

copied in the box under the selected/open item. The user can change the copied text, but the page numbers from the selection will automatically be added to the final report. For example, to complete the guideline item ABSTRACT, the user would select the text of the abstract in the left panel, and that text would be transferred to the ABSTRACT item in the right panel, along with the page number from the original document.

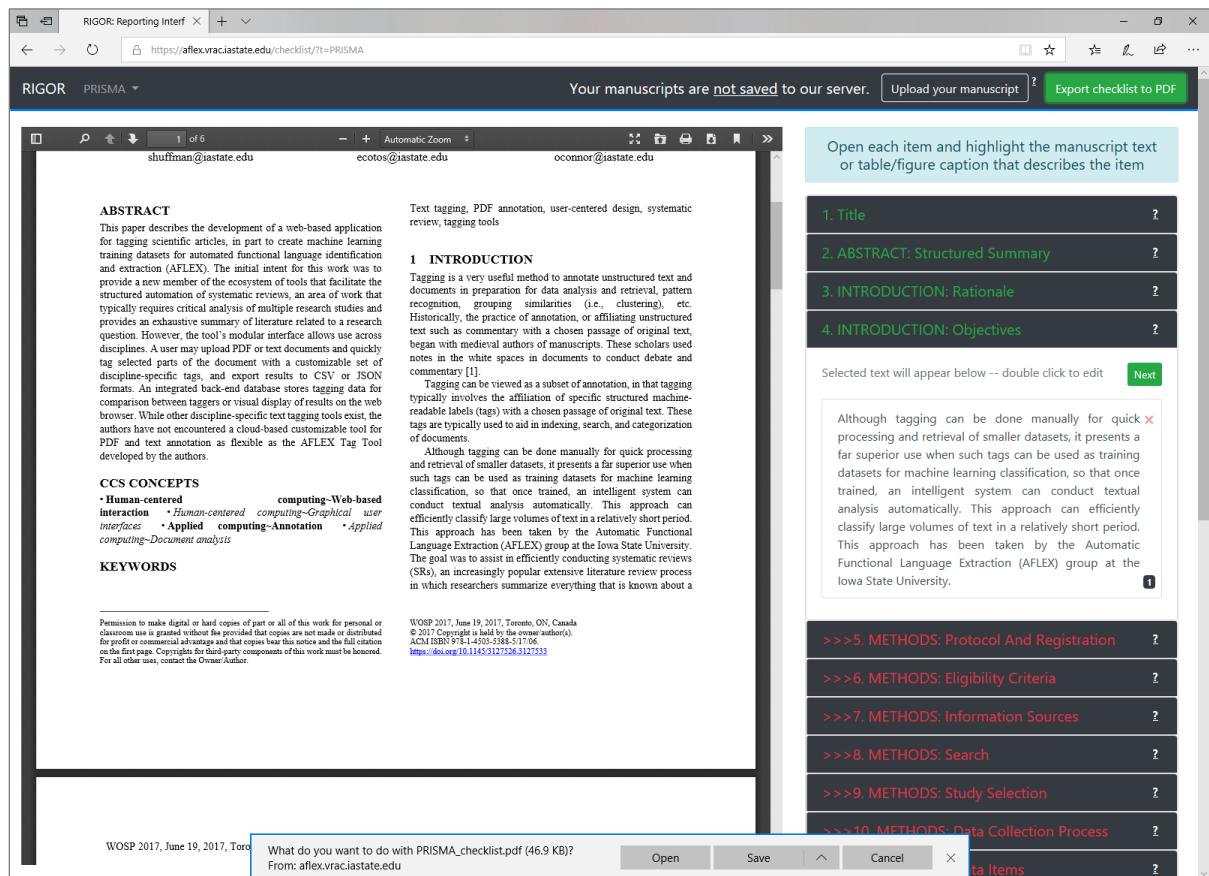
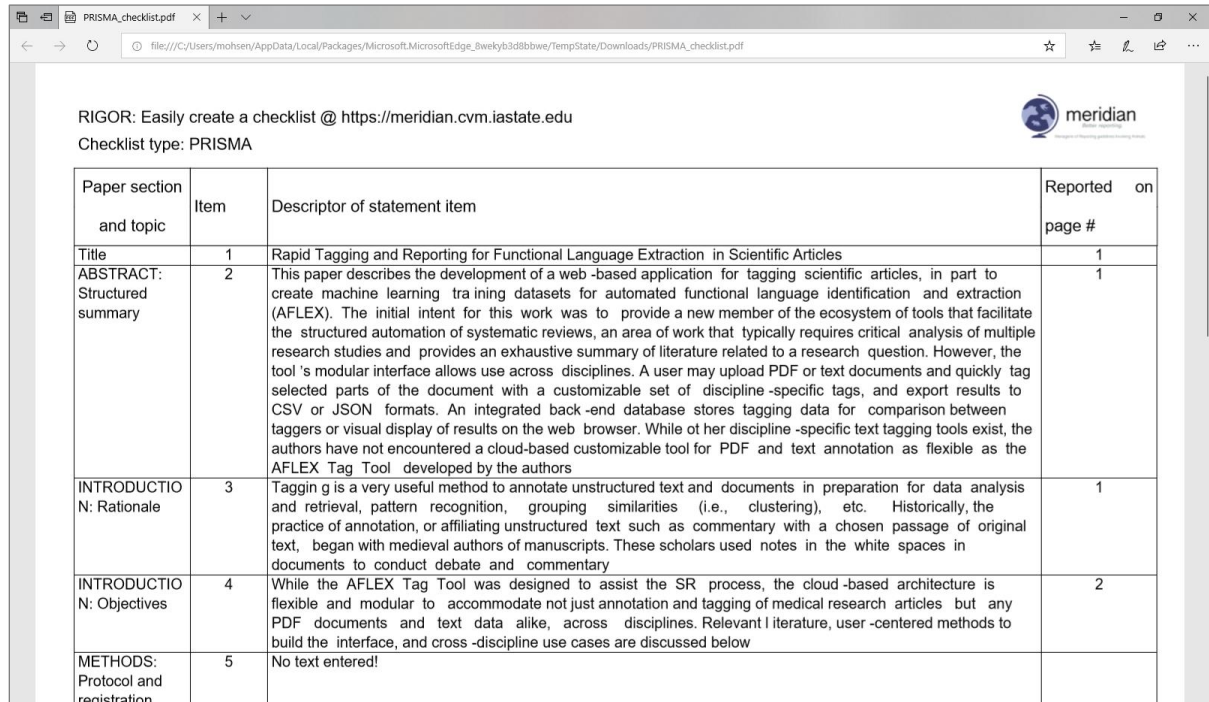


Figure 4-11 A screenshot of the Checklist Tool

After completing the checklist, the user would use the green button on the top right corner that would generate the .pdf output in the selected guideline's layout and structure. A quick validation occurs as the user clicks on export button to let the user know about the checklist items that are completed and the items that are left empty. The titles change color to

red and an indicator appears on the list items to make the empty ones stand out. The output would be downloaded to the user's computer and he would be able to upload it with his manuscript. A screen shot of a part of the output of Checklist Tool is shown in Figure 4-12.



Paper section and topic	Item	Descriptor of statement item	Reported on page #
Title	1	Rapid Tagging and Reporting for Functional Language Extraction in Scientific Articles	1
ABSTRACT: Structured summary	2	This paper describes the development of a web-based application for tagging scientific articles, in part to create machine learning training datasets for automated functional language identification and extraction (AFLEX). The initial intent for this work was to provide a new member of the ecosystem of tools that facilitate the structured automation of systematic reviews, an area of work that typically requires critical analysis of multiple research studies and provides an exhaustive summary of literature related to a research question. However, the tool's modular interface allows use across disciplines. A user may upload PDF or text documents and quickly tag selected parts of the document with a customizable set of discipline-specific tags, and export results to CSV or JSON formats. An integrated back-end database stores tagging data for comparison between taggers or visual display of results on the web browser. While other discipline-specific text tagging tools exist, the authors have not encountered a cloud-based customizable tool for PDF and text annotation as flexible as the AFLEX Tag Tool developed by the authors	1
INTRODUCTION: Rationale	3	Tagging is a very useful method to annotate unstructured text and documents in preparation for data analysis and retrieval, pattern recognition, grouping similarities (i.e., clustering), etc. Historically, the practice of annotation, or affiliating unstructured text such as commentary with a chosen passage of original text, began with medieval authors of manuscripts. These scholars used notes in the white spaces in documents to conduct debate and commentary	1
INTRODUCTION: Objectives	4	While the AFLEX Tag Tool was designed to assist the SR process, the cloud-based architecture is flexible and modular to accommodate not just annotation and tagging of medical research articles but any PDF documents and text data alike, across disciplines. Relevant literature, user-centered methods to build the interface, and cross-discipline use cases are discussed below	2
METHODS: Protocol and registration	5	No text entered!	

Figure 4-12 A screen shot of Checklist Tool output, a PDF file.

File format conversion

The Checklist Tool was designed to facilitate the checklist completion, but it supported only .pdf files at the initial launch. Different journals require different file formats for the submission. Thus, the checklist tool needed to support common file formats that are needed by different journals. After discovering this need, Vlad Sukhoy, a member of AFLEX development team, created a solution for the problem. He used Google Drive Services as a proxy for the Checklist Tool that would convert most document file formats (.doc, .docx, .dot, .html, .odt, .rtf, .txt) to .pdf prior to being rendered in Checklist Tool. Using a third-

party converter module also directs the liability of storing manuscript files to the user's private Google account.

AFLEX Tag Tool Maintenance

To build an application is a complicated and time-consuming task. However, if the code is not well written, then the maintenance can be quite difficult. Anecdotally, developers can often cite from their experience examples of software that were not worth the maintenance and that needed to be rebuilt from the scratch, only because the original developer did not follow best practices in coding.

Documentation

The code documentation was all done inside the code itself as inline or block comments, instead of in separate documents. This approach will help other developers to understand the code more easily and quickly. Also, it did not require a long time and a separate effort to do that. Documentation was done while the application is being developed, ensuring accuracy and efficiency.

CHAPTER 5. DISCUSSION

This chapter is dedicated to discussions on the ATTA development, outcomes, challenges, and limits that the author encountered during the project. During this project, a modular user interface was produced. All the modules use the same database, so the data can easily be exchanged between them. Although each of the UIs were tailored specifically for the intended users, they have other potential applications across disciplines.

Development

There are some special considerations to which the author attended during the development of this project. These considerations are not necessarily common practices in software development, though the author believes that they are valuable to be considered in every software development process.

Sustainable Design In ATTA

The author suggests that people consider sustainability in every job position they are in. However, in traditional software development, sustainability has often been overlooked (Penzenstadler, 2013). Practices that help save the environment are always appreciated. Considering the ubiquitousness of software applications, these practices will have a huge impact on the environment. There are many sustainable activities that could be done during software development, but the following was the one that the author paid attention to during this project.

In general, everything that makes a system work more efficiently can be considered as a sustainability practice. For instance, if an application is optimized in a way that needs less processing power to run, it means that the whole system consumes less energy to produce same results, which leads to saving the environment.

User interface design considerations

In general, monitors consume more power to render light pictures than dark ones. Thus, using darker colors and patterns, if it is possible, can save power while outputting the same results. Throughout this project, the author not only paid attention to these details in user interface design, but also considered people with color blindness. To follow the principles of universal design (Shneiderman, 2000), people with disabilities should always be considered. The author chose colors that are usable for people with color blindness, in the darker part of the color palette. Also, the interface includes an additional text-based indicator beyond color to communicate with the user.

Challenges

In this section the challenges that were faced during this project are discussed. Some of them are general challenges and some of them are specific to tools created with ATTA.

The .pdf File Type

The PDF (portable document format) file format may be used for text, images, and multimedia elements to be presented (Lukan, 2018). However, when it comes to automatic systematic reviews in which the text needs to be read by a machine, it is difficult to extract the text from a .pdf file without any errors. The reason is that first, there are multiple versions of .pdf file types that are different in their internal structure and, second, the text is not stored the way it is rendered (Lukan, 2018). For example, to be able to extract a sentence, one needs to find the position of the letters that formed the sentence and put them together. If a sentence spans multiple pages in a .pdf file, for example, the first section and the second section of the sentence are likely stored in very different places internally within the file.

Using a client-side library like PDF.js helped this process significantly as the library not only renders the .pdf file as HTML, but also has some functions that extract a selected

part of the text from that .pdf file. Given that this process is still not robust, there are some heuristics that are implanted in the ATTA code to take care of problems like line breaks, spaces, etc. It must be mentioned that there is still a small chance that when a user selects a text excerpt, the output could be very different. Also, some of the .pdf files are not compatible with the PDF.js library and as a result, they cannot be rendered at all.

Evaluation

The evaluation criteria for these types of tools were mentioned in Chapter 1 of this document. The author evaluated the AFLEX tag tools in comparison with other tools that were discussed in Chapter 2 against the criteria.

Table 5-1 A comparison between AFLEX tag tools and similar tools in the literature. N/A indicates that not enough information was available for evaluation.

	Multi-user support	User centric UI design	Conflict resolution system	Input file type support	Output file types	Data storage type	Extensible / support plug-ins
AFLEX tag tools	YES	YES	YES	.pdf, .doc, .docx, .dot, .html, .odt, .rtf, .txt	.csv, .xml, .json	separate from file	YES
BRAT	YES	N/A	YES	.txt	N/A	separate from file	NO
Callisto Annotation Workbench	NO	N/A	NO	.txt	.pdf, figures	separate from file	YES
Adobe Acrobat	N/A	N/A	NO	.pdf	.pdf	In the file	NO

Future Work

Machine Learning Implementation

As it is noted before, ATTA is a part of the project AFLEX which involves natural language processing using machine learning algorithms. The tools resulting from ATTA helped collect data for the purpose of AFLEX. However, currently the machine learning part is not finalized and still is under development. The machine learning part can be developed in parallel to the data collection. The important point is that the data should be stored and exchanged in the way that the machine learning components needs it, so that ATTA serves as a smooth pipeline to future text processing tools.

Data exchange pipeline

In this document, it has been mentioned multiple times that all the user interfaces of ATTA tools share the same database. Thus, there is no problem of exchanging data between those interfaces. However, to be able to export data to be used by other applications such as machine learning algorithms, there must be a pipeline that would enable applications to connect and exchange data. The most common way is to create a webservice that has access to the database and would expose some end points for other applications to connect to them.

There are standard ways of creating a web service. The author recommends creating a RESTful web service (Richardson & Ruby, 2008) that would be called by the machine algorithm for the data exchange. A diagram of this architecture is shown in Figure 5-1. It also should be noted, as it is mentioned in the diagram, that the RESTful service will only support read operations, meaning that it would not allow changes to the data.

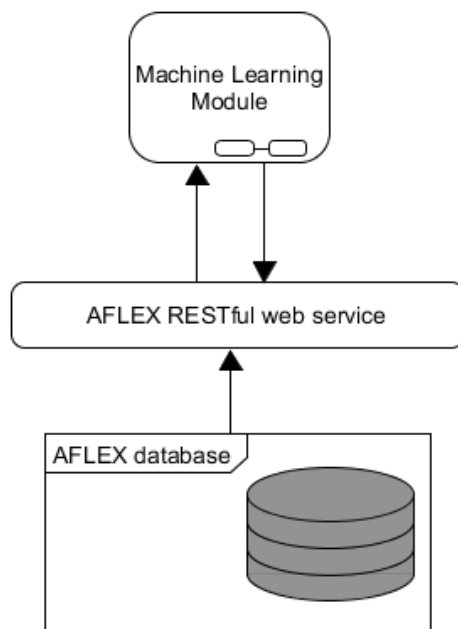


Figure 5-1 A diagram of the communication between an ATTA tool and an AFLEX ML module.

AI-Assisted Checklist Tool

One of the outcomes of this project is the Checklist Tool that is used to help completing the checklists that are required by some journals. Currently the process is manual. However, developing an AI-assisted system that would fill out the checklist items automatically is not unattainable. In addition to the need for a trained machine learning algorithm, there must be an abstraction layer above all the guidelines from different journals. E.g., one guideline might call the first item as "title" and the second item "abstract," but another guidelines might call the same concepts "title and abstract, item 1a" in a single checklist item. An abstraction layer would recognize and align these different concepts underlying the items. If the AI is trained based on the concepts, not the titles of items, then it could become journal-independent, allowing it to potentially be used across journals or even other domains and languages.

UX Research on Tool Usability

The author designed and created all the tools based on users' needs, with user experience (UX) and software engineering principles in mind. However, the success and efficiency of the tools has yet to be verified. In human computer interaction, there are methods to evaluate user experience. It is recommended to time users as they tag the documents manually and compare these data with the time required when they accomplish the same task using ATTA tools. The timing data of users' actions in Tag Tool 1, for example, is already collected and stored in the AFLEX database.

REFERENCES

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. In *Software Engineering, 2003. Proceedings. 25th International Conference on* (pp. 244–254).

Adeva, J. J. G., Atxa, J. M. P., Carrillo, M. U., & Zengotitabengoa, E. A. (2014). Automatic text classification to support systematic reviews in medicine. *Expert Systems with Applications*, 41(4), 1498–1508.

Albert, W., & Tullis, T. (2013). *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes.

Banko, M., & Brill, E. (2001). Scaling to Very Very Large Corpora for Natural Language Disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics* (pp. 26–33). Stroudsburg, PA, USA: Association for Computational Linguistics. <https://doi.org/10.3115/1073012.1073017>

Baxter, K., Courage, C., & Caine, K. (2015). *Understanding your users: A practical guide to user research methods*. Morgan Kaufmann.

Boyanov, N. (2005). UMS - User Management System. Retrieved from <http://ums.sourceforge.net/home/>

Bui, D. D. A., Del Fiore, G., Hurdle, J. F., & Jonnalagadda, S. (2016). Extractive text summarization system to aid data extraction from full text in systematic review development. *Journal of Biomedical Informatics*, 64, 265–272.

Carlile, P. R. (2002). A pragmatic view of knowledge and boundaries: Boundary objects in new product development. *Organization Science*, 13(4), 442–455.

Cotos, E. (2016). Computer-assisted research writing in the disciplines. *Adaptive Educational Technologies for Literacy Instruction*, 225–242.

Cotos, E., Huffman, S., & Link, S. (2015). Furthering and applying move/step constructs: Technology-driven marshalling of Swalesian genre theory for EAP pedagogy. *Journal of English for Academic Purposes*, 19, 52–72.

Cotos, E., Huffman, S., & Link, S. (2017). A move/step model for methods sections: Demonstrating rigour and credibility. *English for Specific Purposes*, 46, 90–106.

Day, D. S., McHenry, C., Kozierok, R., & Riek, L. (2004). Callisto: A Configurable Annotation Workbench. In *LREC*.

Dwan, K., Altman, D. G., Arnaiz, J. A., Bloom, J., Chan, A.-W., Cronin, E., ... others. (2008). Systematic review of the empirical evidence of study publication bias and outcome reporting bias. *PloS One*, 3(8), e3081.

Fraser, G., & Arcuri, A. (2011). EvoSuite: Automatic Test Suite Generation for Object-oriented Software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (pp. 416–419). New York, NY, USA: ACM. <https://doi.org/10.1145/2025113.2025179>

Gough, D., & Elbourne, D. (2002). Systematic research synthesis to inform policy, practice and democratic debate. *Social Policy and Society*, 1(3), 225–236.

Harmsze, F.-A. P. (2000). A modular structure for scientific articles in an electronic environment.

Kando, N. (1999). Text structure analysis as a tool to make retrieved documents usable. In *Proceedings of the 4th International Workshop on Information Retrieval with Asian Languages* (pp. 126–135).

Kaur, R., Sidhu, P., & Singh, S. (2016). What failed BIA 10-2474 Phase I clinical trial? Global speculations and recommendations for future Phase I trials. *Journal of Pharmacology & Pharmacotherapeutics*, 7(3), 120–126. <https://doi.org/10.4103/0976-500X.189661>

Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

Loeliger, J., & McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development*. “O’Reilly Media, Inc.”

Lukan, D. (2018). PDF File Format: Basic Structure.

Lunn, K. (2003). Software Development Life Cycle. In *Software development with UML* (pp. 53–68). Springer.

MacCormick, J. (2011). *Nine algorithms that changed the future: The ingenious ideas that drive today’s computers*. Princeton University Press.

Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27, 504–518.

Marshall, I. J., Kuiper, J., & Wallace, B. C. (2016). RobotReviewer: evaluation of a system for automatically assessing bias in clinical trials. *Journal of the American Medical Informatics Association*, 23(1), 193–201. Retrieved from <http://dx.doi.org/10.1093/jamia/ocv044>

- Matthews, G., Davies, D. R., Stammers, R. B., & Westerman, S. J. (2000). *Human performance: Cognition, stress, and individual differences*. Psychology Press.
- Moher, D., Liberati, A., Tetzlaff, J., & Altman, D. G. (2009). Preferred reporting items for systematic reviews and meta-analyses: the PRISMA statement. *Annals of Internal Medicine*, 151(4), 264–269.
- Moniruzzaman, A. B. M., & Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *ArXiv Preprint ArXiv:1307.0191*.
- O'Connor, A. M., Totton, S. C., Cullen, J. N., Ramezani, M., Kalivarapu, V., Yuan, C., & Gilbert, S. B. (2018). The study design elements employed by researchers in preclinical animal experiments from two research domains and implications for automation of systematic reviews. *PloS One*, 13(6), e0199441.
- O'Connor, A. M., Tsafnat, G., Gilbert, S. B., Thayer, K. A., & Wolfe, M. S. (2018). Moving toward the automation of the systematic review process: a summary of discussions at the second meeting of International Collaboration for the Automation of Systematic Reviews (ICASR). *Systematic Reviews*, 7(1), 3. <https://doi.org/10.1186/s13643-017-0667-4>
- Okumura, Y. (2016). Reducing Research Waste Through Good Reporting Practices. *Journal of Epidemiology*, 26(8), 397–398. <https://doi.org/10.2188/jea.JE20160105>
- Penzenstadler, B. (2013). What does Sustainability mean in and for Software Engineering. In *Proceedings of the 1st International Conference on ICT for Sustainability (ICT4S)*.
- Ramezani, M., Kalivarapu, V., Gilbert, S. B., Huffman, S., Cotos, E., & O'Conner, A. (2017). Rapid Tagging and Reporting for Functional Language Extraction in Scientific Articles. In *Proceedings of the 6th International Workshop on Mining Scientific Publications* (pp. 34–39).
- Reller, T. (2016). Elsevier publishing – a look at the numbers, and more. Retrieved from <https://www.elsevier.com/connect/elsevier-publishing-a-look-at-the-numbers-and-more>
- Richardson, L., & Ruby, S. (2008). *RESTful web services*. “O'Reilly Media, Inc.”
- Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *Unified modeling language reference manual, the*. Pearson Higher Education.
- Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35(3), 8–13.

Schulz, K. F., Altman, D. G., & Moher, D. (2010). CONSORT 2010 statement: updated guidelines for reporting parallel group randomised trials. *BMC Medicine*, 8(1), 18.

Shneiderman, B. (2000). Universal usability. *Communications of the ACM*, 43(5), 84–91.

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., & Tsujii, J. (2012). BRAT: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics* (pp. 102–107).

The EQUATOR Network. (n.d.). The EQUATOR Network. Retrieved from <http://www.equator-network.org/>

Thomas, J., McNaught, J., & Ananiadou, S. (2011). Applications of text mining within systematic reviews. *Research Synthesis Methods*, 2(1), 1–14.

Tranfield, D., Denyer, D., & Smart, P. (2003). Towards a methodology for developing evidence-informed management knowledge by means of systematic review. *British Journal of Management*, 14(3), 207–222.

Tsafnat, G., Glasziou, P., Choong, M. K., Dunn, A., Galgani, F., & Coiera, E. (2014). Systematic review automation technologies. *Systematic Reviews*, 3(1), 74.

Uman, L. S. (2011). Systematic reviews and meta-analyses. *Journal of the Canadian Academy of Child and Adolescent Psychiatry = Journal de l'Académie Canadienne de Psychiatrie de l'enfant et de l'adolescent*, 20(1), 57–59. Retrieved from <https://www.ncbi.nlm.nih.gov/pubmed/21286370>

Von Elm, E., Altman, D. G., Egger, M., Pocock, S. J., Gøtzsche, P. C., Vandenbroucke, J. P., ... others. (2007). The Strengthening the Reporting of Observational Studies in Epidemiology (STROBE) statement: guidelines for reporting observational studies. *PLoS Medicine*, 4(10), e296.

Wallace, B. C., Kuiper, J., Sharma, A., Zhu, M., & Marshall, I. J. (2016). Extracting PICO sentences from clinical trial reports using supervised distant supervision. *The Journal of Machine Learning Research*, 17(1), 4572–4596.

Wolfe, J. (2002). Annotation technologies: A software and research review. *Computers and Composition*, 19(4), 471–497. [https://doi.org/10.1016/S8755-4615\(02\)00144-5](https://doi.org/10.1016/S8755-4615(02)00144-5)

Zhi, J., Garousi-Yusifoglu, V., Sun, B., Garousi, G., Shahnewaz, S., & Ruhe, G. (2015). Cost, benefits and quality of software development documentation: A systematic mapping. *Journal of Systems and Software*, 99, 175–198.