Lab 2: A^3 framework Product Specification Outline

Aaron Berman

CS 411W: Workforce Development II

Thomas J. Kennedy

Janet Brunelle

June 24, 2020

Version 1

Table of Contents

List of Figures

List of Tables

1. Introduction

The Coronavirus Disease 2019 (COVID-19) response of moving education to the on-line format highlights the breakdown of traditional knowledge management solutions. The lack of knowledge management, in a traditional environment, is due to the fractured and specialized approach that is utilized in a non-centralized environment. Traditional models of knowledge management include both artifacts and personal knowledge; artifacts being the codified knowledge, written or presented on media, and personal knowledge, being experience or understanding of a subject (Carrol et al., 2003). The scope of this paper is limited to artifacts, with the assumption that personal knowledge will be codified in some form later. Davenport et al. (1997), suggests that traditional models have no central framework to aggregate and archive artifacts. One of the main obstacles to a centralized knowledge management solution is artifacts are individualized and created to the specialization of the creator, according to Kennedy (T. Kennedy, personal communication, February 12, 2020.) A byproduct of isolation by specialization and a non-centralized environment is the inability to track changes over time, which leads to more artifacts becoming abandoned or lost. With the loss of aggregation and archival, artifacts may become abandoned or lost due to reassignment of responsibilities of the creator (J. Brunelle, personal communication, March 2, 2020.) Abandonment and loss of artifacts create a demand for duplication of work for educators and researchers and limit the number of resources educators, researchers, and students have access to when researching a topic of interest or need.

1.1. Purpose

A^3 (A cubed) framework will allow for the aggregation and archival of artifacts for educators, researchers, and students by overcoming the challenges of individualization and location in an academic environment while attempting the reduce the number of artifacts which are lost or abandoned. A^3 framework will be a configurable and extensible knowledge management framework which will be based on the core functionality of normalization, comparison using the Diff command, and storage using a centralized database/repository system. Authentication and access control will be employed for data security. Users will be able to interact with A^3 framework utilizing either a Command Line Interface (CLI) or a Graphical User Interface (GUI). The user interface will connect to the database via RESTful APIs.

1.2. Scope

A^3 framework will be a centralized database with repositories for each user. Individual repositories will allow users to keep artifacts in a personal repository. A^3 framework will normalize artifacts to a markdown format, which is both compressible for storage and convertible to other artifact extension types. With normalization a line-by-line comparison of the artifacts' contents will be achievable, with a highlighted report generated and displayed to the user. The normalization process to markdown allows artifacts to be stored in a binary large object (BLOB) directly within a user's repository. Any artifact which cannot be normalized will be stored alongside the repository with a pointer to the artifact's location. This approach will allow A^3 framework to achieve a more complete archival process and allow comparisons of artifact attributes which will be useful in generating reports of how each artifact has changed over time. The normalization and centralization of artifacts will allow specialized resources of one individual to be reformatted and utilized by another, while maintaining the contents and

research efforts of the original. Tagging of artifacts will allow users to search the repositories to filter through the artifacts to find the relevant artifacts to their subject of concern. This is advantageous for in both academic and research environments where specialization occurs frequently, however, cross-specialization or cross-disciplinary research may be needed to understand a problem's scope. The GUI would extend the CLI functionality by showing a graphical representation of functions e.g., the Diff command and notifications. Notifications, filtering parameters, and bookmarks, while available in the CLI format, will have a meaningful impact for both guest and student accounts providing an intuitive layout with minimal training required for use. Notifications will be user alerts to allow the database to request intervention on an artifacts behalf when an artifact meets certain criteria such as potential abandonment or the artifact has been updated. Notifications will allow the user to maintain artifacts in a manageable and informed way.

A^3 framework is designed to be used primarily in a University environment, with a specific emphasis on educators, researchers, and students. As such the A^3 framework permission levels have been designated: guest, student, faculty, administrator, and tester, for development purposes only. A^3 framework prototype will have limited set of permission levels which showcase the core functionality of normalization, comparison, and centralized storage. Permission levels designated student and administrator are by design a subset of the faculty and tester permission levels respectively; therefore, the student and administrator permission levels will be eliminated for the prototype. A^3 framework prototype will implement limited tagging functionality, thereby, reducing development resources during the prototyping phase. Tagging of artifacts, during the prototype phase, will be handled through the database with manual entry.

[This space intentionally left blank.]

1.3. Definitions, Acronyms, and Abbreviations

Aggregate: Data that is composed of smaller pieces that form a larger whole.

Algorithm: Set of instructions designed to perform a specific task.

Angular: A framework for dynamic web apps. Allows for the use of HTML as a template language.

Application Programming Interface (API): Set of functions and procedures allowing the creation of applications that access features of an operating system, applications, etc.

Archive: Contains multiple files and/or folders. May be created by several different utilities and may be saved in different formats.

Artifact: Combination of arte, “by skill”, and factum, “to make”. A file or document.

Backlink: A hyperlink that links from a web page, back to your own web page or website.

Blackboard: A tool that allows faculty to add resources for students to access online.

Centralized: Type of network where all users connect to a central server.

Course Websites from Markdown (CoWeM): A system for building course websites, including notes, slides, and organizational pages, from Markdown documents.

Cascading Style Sheet (CSS): Used to format the layout of web pages. Defines text styles, table sizes, among other things that previously could only be defined in HTML.

Database: Collection of information, that is organized for rapid search and retrieval.

Data Loss: An instance in which information is destroyed by failures or neglect.

Diff: A line by line comparison of normalized artifacts.

Docker: Tool to create, deploy, and run applications by using containers. Allow developers to package up an application, with all parts needed, to be deployed in one package.

Export: Taking data from one program or computer to another.

GitLab: Used to provide internal management of git repositories. Is a self-hosted Git-repository management system that keeps the user code private.

Graphical User Interface (GUI): User interface that contains graphical elements. Examples include windows, icons and buttons.

Hypertext Markup Language (HTML): A language used to create web pages. “Hypertext” refers to hyperlinks in a page, and “Markup language” refers to the way tags are used to define page layout.

Hyperlink: An element that links to another file or object.

JavaScript (JS): A language used in web development. While influenced by Java, it’s syntax is more similar to C.

Knowledge Management: The management process of creating, capturing, sharing, retrieving, and storing data, information, knowledge experiences and skills by using appropriate information and network technology.

Markdown: A markup language that can be used to format plain text. Can be converted into another language.

Markup: A language that uses tags to define elements within a document.

MySQL: Open source SQL database management system. Developed and distributed by Oracle Corporation.

Normalization: Converting ingested objects into a small number of pre-selected formats.

Python: An interpreted, object-oriented language.

Personal Learning Environment (PLE): An interface used in flexible online courses. Designed by ODU's Center for Learning and Teaching.

pydoc: Automatically generates documentation from Python modules. Can be presented as pages of text on the console, served to a web browser, or saved to HTML files.

Pylint: A Python static code analysis tool. Looks for programming errors and warnings from within the code, as well as from an extensive configuration file.

React: A JavaScript library that is used to create User Interfaces for web applications.

reStructuredText: A plaintext markup syntax and parser system. Useful for in-line program documentation.

Secure File Transfer Protocol (SFTP): Secure version of File Transfer Protocol. Facilitates data access and data transfer over a Secure Shell data stream

Sphinx: A Python documentation generator. Converts reStructuredText files into HTML websites and other formats.

Tags: Is a keyword or term assigned to a piece of information.

tox: Aims to automate and standardize testing in Python. Is a generic virtualenv management and test command line tool.

Visual Studio Code: A source code editor that runs on Mac, Linux, and Windows.

[This space intentionally left blank.]

1.4. References

Blackboard Archive Extractor. (2016, December 15) cs.odu.edu. Retrieved March 10, 2020, from https://www.cs.odu.edu/~cpi/old/411/crystals17/.

Berman, A. (2020, June 15) *Lab 1: A^3 framework Product Description.* cs.odu.edu. Retrieved June 20, 2020, from https://www.cs.odu.edu/~411crystal/labs.html#berman

Carroll, J., Choo, C. W., Dunlap, D., Isenhour, P., Kerr, S., MacLean, A., & Rosson, M. (2003). Knowledge Management Support for Teachers. Educational Technology Research and Development, 51(4), 42-64. www.jstor.org/stable/30221184

Davenport, T., Long, M. & Beers, M.. (1997). *Building Successful Knowledge Management Projects* [Working Paper]. Retrieved March 8, 2020, from https://www.researchgate.net/publication/200045855_Building_Successful_Knowledge_Management_Projects.

Document Management Software | eFileCabinet. (2020). eFileCabinet. Retrieved February 20, 2020, from https://www.efilecabinet.com.

Domes, S. (2017). *Progressive Web Apps with React: Create lightning fast web apps with native power using React and Firebase.* Packt Publishing Ltd.

File Sharing and Sync For Education, Schools and Universities - FileCloud. (2020). FileCloud. Retrieved February 20, 2020, from https://www.getfilecloud.com/file-sharing-and-sync-for-education/.

GitHub Features: The right tools for the job. (2020). GitHub. Retrieved March 10,il 2020, from https://github.com/features#team-management.

Kennedy, T. (2020, January 21). *Home · Wiki · Thomas J. Kennedy / cs-roars-proposal*. GitLab. Retrieved 26 April 2020, from https://git-community.cs.odu.edu/tkennedy/cs-roars-proposal/-/wikis/home.

Nvlpubs.nist.gov. (n.d.). *Glossary of Key Information Security Terms.* From https://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf.

MacFarlane, J. (2006). *Pandoc - About pandoc*. Pandoc.org. From https://pandoc.org/index.html.

Tsapps.nist.gov. (2020). *Data Loss Prevention.* From https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=904672.

Xie, I., & Matusiak, K. K. (2016, July 29). Digital preservation. *Science Direct* (255-279). Retrieved March 10, 2020, from https://www.sciencedirect.com/science/article/pii/B9780124171121000090

Zeil, S. (2019, December 26). *Building the Website*. cs.odu.edu. Retrieved 26 April 2020, from https://www.cs.odu.edu/~zeil/cowem/Public/buildingTheWebsite/index.html.

Zeil, S. (2020, January 21). *zeil / CoWeM - Course Websites from Markdown*. GitLab. From https://git-community.cs.odu.edu/zeil/Course_Website_Management.

1.5. Overview

This product specification provides a high-level description of A^3 framework, with emphasis on major functional components and why they were selected for the real-world product and which components, if any, have been changed for the prototype. The information provided in the remaining sections will also give a detailed description of the functional capabilities which are necessary for A^3 framework or A^3 framework prototype, in an academic or research environment, and any performance requirements identified by the customer or through standard testing of products that are comparable to a component of A^3 framework. For example, web page render times, while web page rendering is not A^3 framework's goal, a component of A^3 framework is web based, therefore, rendering speeds of web pages should be consistent with other web page rendering averages. This product specification will include a detailed description the parameters that will be used to control, manage, or establish each function and the performance characteristics of the function in terms of output, display, and user interaction.

2. General Description

A^3 framework will encompass a CLI, GUI, web scraper, and database with repositories deployed via Docker containers. While the algorithms traverse both front and back end, the most resource intensive processes will be handled by the back end, which includes the normalization and filtering of artifacts. This will enable A^3 framework to provide a similar user experience regardless of customer location. Figure 1 illustrates the major functional components of A^3 framework.

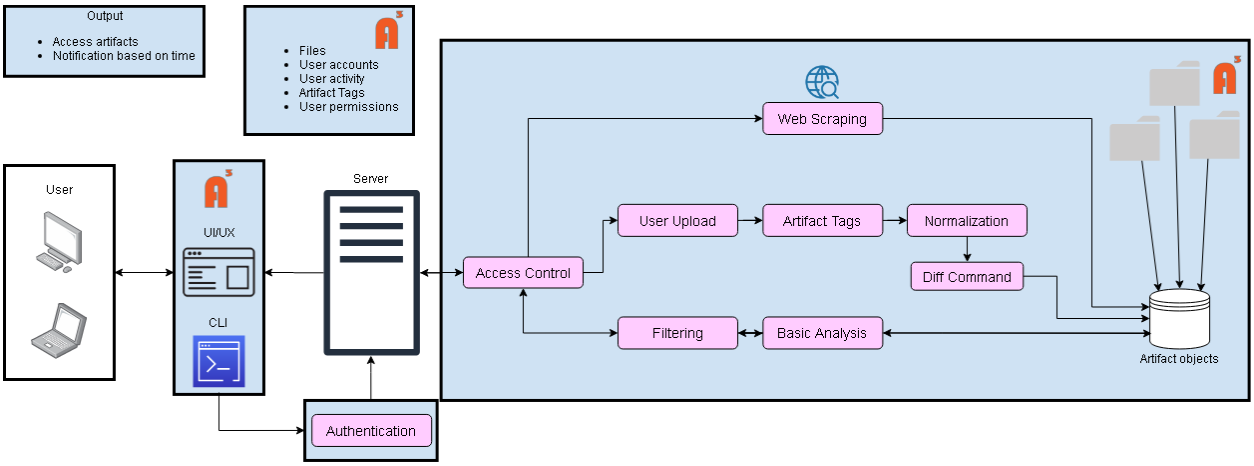

Figure 1: *Major functional components of A^3 framework.*

A^3 framework will have two user interfaces, GUI and CLI, which will be utilized by the user to interact with A^3 framework's database for managing individual repositories. The GUI will be implemented using HTML, CSS, and the JS framework React. The command line and underlying code will be written using Python 3.8 or newer. A^3 framework will use pydoc and Sphinx for documentation management and Pylint and pydocstyle (formerly PEP 8) for coverage and code analysis. This will provide the user with an interface that integrates the front-end (GUI and CLI) and back-end (database/repository) using the REST API. A^3 framework's database will run on a virtual machine instance, running an Ubuntu Linux distribution, on a server with docker containerization, which will be handled with Docker and Docker compose.

A^3 framework will be developed utilizing the GitLab code repository and Visual Studio Code IDE. With configuration management being handled by tox and virtualenv. The database will use MySQL, with the potential to migrate to MongoDB as a potential better database solution to the A^3 framework.

2.1. Prototype Architecture Description

A^3 framework prototype will implement key algorithms from the final product, which show an innovative/novel approach to knowledge management. The prototype will demonstrate the retrieval, normalization, storage, and comparison of artifacts in a measurable and reportable way. The Prototype will have limited tagging and filtering algorithms for demonstration purposes. A^3 framework prototype will utilize all hardware and software specifications of the real-world product as shown in Figure 2.

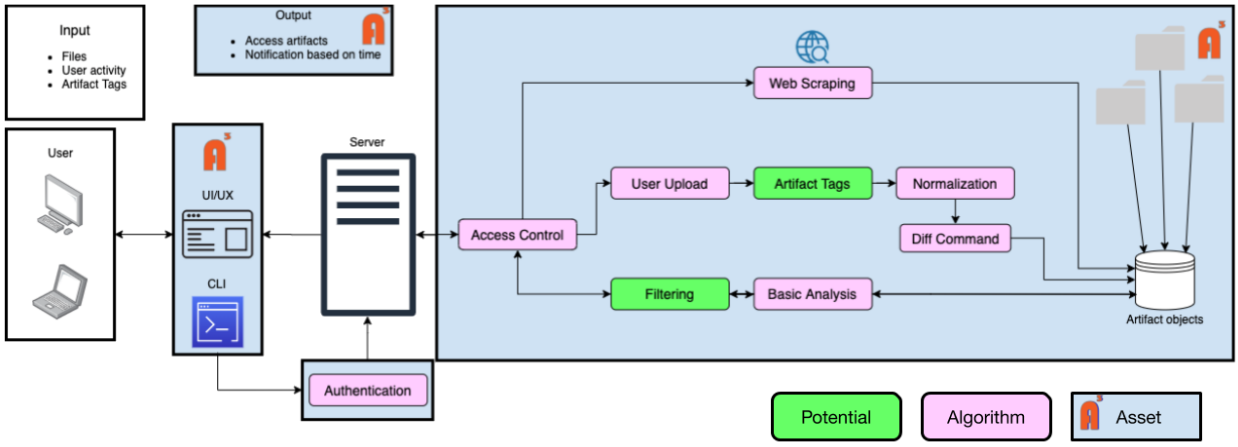

Figure 2: *Prototype major functional component diagram.*

A^3 framework prototype is comprised of the following major components:

- GUI: provides a web-based application interface which will allow a user to login, filter artifacts, retrieve an artifact, upload or update an artifact, and request and view reports about an artifact.
- CLI: provides an interface by which a user can perform the same activities as the GUI interface with the added functionality of being able to be added to new or existing automation activities. Examples include but are not limited to scripting and Git hooks.
- Database: provides a storage and organization medium by which artifacts and information can be stored/requested using SQL. The database provides an organization and request scheme which allows a user to store/request an individual artifact using constraints which allow the user to access an artifact regardless of catalog size.
- Server: provides a two-way communication to the GUI/CLI to the database without the risk of corrupting the database. This is done using RESTful APIs which sanitizes the inputs of a user while allowing the user to utilize a more natural language for database queries. The server will also handle algorithms such as web scraping, basic analysis, preprocessing of Diff commands, etc.

A^3 framework prototype will be developed using Visual Studio Code for use across platforms by utilizing Python 3.8 or newer for the CLI and JavaScript/React for the web-based GUI. A^3 framework will utilize a network connection for web scraping and connection to the A^3 framework database and repositories. Filtering and tagging of artifacts will be limited to manual input only as the key features which will be highlighted are the normalization, storage, and comparison of artifacts.

2.2. Prototype Functional Description

A^3 framework prototype will demonstrate aggregation and normalization of electronic artifacts for archival can be done systematically with reportable metrics. A^3 framework prototype will have some limited functionality in comparison to the real-world product as outlined in Table 1.

Feature/Capabilities Comparison Chart		
Feature/Capability	**Real World**	**A³ Prototype**
Database Storage	X	X
Graphical User Interface	X	**Limited**
Command Line Interface	X	X
User Authentication	X	**Limited**
Access Control	X	X
Artifact Upload	X	X
Repository Creation	X	X
Artifact Normalization	X	X
Artifact Comparison	X	X
Artifact Update	X	X
Artifact/Repo Deletion	X	
Web Scraping	X	**Limited**
Artifact Charge Record	X	X
Artifact Exporting	X	X
Artifact/Repo Searching	X	**Limited**
Artifact Contributor List	X	
Artifact/Repo Sharing	X	
Artifact/Repo Comments	X	

Table 1: *A^3 framework real world product versus prototype comparison.*

A^3 framework prototype will store artifacts in a communal repository of artifacts. Users at the Guest level will only be able to view the contents of public artifacts and Faculty level users will have access to all artifacts, thereby, demonstrating RBAC. User authentication will be a username/password combination, allowing Faculty or Tester level users to authenticate to A^3 framework's database and repositories. Both RBAC and authentication will allow A^3 framework to demonstrate the implementation of security controls for both confidentiality and integrity of data stored within A^3 framework prototype. A fully featured CLI will be developed as well as a

limited GUI overlay. The limited GUI will allow for sign on, artifact upload, limited filtering, side-by-side difference comparison, and viewing of the artifact list. The GUI will be developed for a normalized windowed experience for a subset of users and the CLI will be developed for use in the automation of tasks, by scripting or other frameworks, as well as the subset of users working in a non-graphical environment. A^3 framework prototype will allow for direct resource updating, with previous versions becoming automatically archived, and updating through web scraping. The use of resource updating and archival will allow A^3 framework to track changes over time and report measurable metrics for use in other research areas. Authentication will be a single sign on (SSO) implementation with extensibility to outside authentication services, such as the Monarch Identification and Authorization Service (MIDAS.) Once authorized by the authentication agent for access to the server, A^3 framework will check a user's access level using Role Based Access Control (RBAC). The user roles of guest and faculty will be defined, with the tester role used only in the development and testing of A^3 framework. All user roles will have access to the Filtering algorithm, which allows search parameters, such as tags, to return artifact ids from repositories in the database, allowing the user to select the artifact that fits the user's needs and allowing inspection of the contents. Faculty and above user roles will allow for Web Scraping, to upload artifacts from a web host, and local source uploads. Web Scraping enables the automation and/or batching of artifacts helping to mitigate abandonment and loss of artifacts from traditional models of knowledge management. User uploads encompasses both the first time upload and update processes by performing a tag check phase during the upload algorithm which determines if a change record exists and creating a new entry in the database field with the change recorded, by default if no change has occurred or a new artifact has been uploaded a "No Change" is recorded. Every artifact will be compared to a list of normalizable filetypes ,e.g., portable document format (PDF), Microsoft Word document (DOC and DOCX), Microsoft Power Point (PPT and PPTX), Hypertext Markup Language (HTM and HTML), and Open Document text (ODT), if an artifact cannot be normalized then a simplified Diff command will be performed which only compares the artifact's attributes, e.g., artifact size, artifact creation date, and artifact name. If an artifact can be normalized, then the file will be converted to markdown format. The converted artifact will then be compared to the previously stored artifact with the Diff command. The result of the Diff command will then be displayed to the user. The storage of the Diff command output in the change record will allow the user to track changes to an artifact over time. Basic analysis will be a return only algorithm which will inspect an artifact at the file attribute level, for faculty and above users, to view details on artifacts they need more information about, e.g., owner, creation date, or size. This will allow for maintainability of the knowledge management database as well as attribution for cited works.

2.3. Extended Interfaces

A^3 framework prototype will not interface with outside services; however, through the use of Docker-compose, docker containers, and Python frameworks, such as Flask and mysql-connector, A^3 framework will interface between client side GUI/CLI to the A^3 framework server and from the A^3 framework server to the A^3 framework database. Additional Python frameworks, such as BeautifulSoup4, will allow the interface and retrieval of web hosted artifacts.

2.3.1. Hardware interfaces

N/A

2.3.2. Software Interfaces

Docker: is required on the for deployment of a new A^3 framework database. Docker may be used to deploy the React web application and CLI.

Docker-compose: Is required for deployment of a new A^3 framework database, as it deploys both the server and database in docker containers with customized requirements for each.

2.3.3. User Interfaces

Monitor: for display and interaction with A^3 framework have no minimum specifications.

Keyboard: for data entry.

2.3.4. Communication Protocols and Interfaces

TCP/IP: is the protocol used to connect the GUI/CLI to the server, there is no minimum requirement.

HTTPS: is the end to end connection type for sending user input and database responses.

3. Specific Requirements

3.1. Functional Requirements

3.2. Performance Requirements

3.3. Assumptions and Constraints

3.4. Non-functional Requirements

Appendix.