High Performance XML/XSLT Transformation Server

Spring 2017 Final Project Report

Zixun Lu (luzi), Shuai Peng (pengs), Elijah Voigt (voigte)

OSU CS Senior Capstone 2016-2017

June 8, 2017

Abstract

The Oregon State University Capstone course, in collaboration with the Apache Software Foundation, worked to create a high performance web-available XML document transformer. The end result is a simple XML document transformer accessible through a web-interface, backed by an in-memory caching system and multi-threaded job processor.

CONTENTS

1 Introduction

The *High Performance XML/XSLT Transformation Server* project was proposed by Steven Hathaway in affiliation with the Apache Software Foundation. The original task was described as "Create a high-performance XML/XSLT document transformation server to perform large and repetitive document transformation tasks in a timely manner. The source libraries will be the Apache Xerces-C and Apache Xalan-C products. The server should also include support ICU (International Components for Unicode) in XML parsers and document serialization." The original deliverables were "[...] a client program that can demonstrate several document transformations using the services of an XML/XSLT transformation server."

An XML transformer is an application which takes in two or more structured input files, specifically XML files and XSLT (XML Style Sheets), processes the transformations specified in the XSLT file, and outputs the resulting XML file. For the purposes of this project it is not entirely important to understand the underlying transformations taking place, since the objective of the project was to implement a web service using existing [Apache maintained] libraries and tools, not to re-implement an XML transformer library.

The application's importance comes mainly from large enterprise needs for XML document transformation. Many organizations depend on XML data transformations for a variety of tasks, but tend to perform these tasks on local hardware. By creating an Open Source application which can offload this computation, employees can get more work done (A) not installing an XML transformer application locally and (B) not using their local machine to carry out computation heavy XML transformations.

Steven played a role in the development of the application, making himself available for meetings but not actively mentoring the development team. He did explicitly make himself available for mentor ship if members of the development team needed.

1.1 Development team

The development team consisted of Zixun Lu, Shuai Peng, and Elijah C. Voigt. All three students were seniors in the Computer Science program at OSU. Zixun and Shuai were exchange students from China; Elijah was a local Oregonian.

2 Glossary

3 References

4 Project requirements

Before development can take place the team responsible for a project collects the necessary requirements for a project and criteria for success. These are what the client ought to "grade" the final product against to judge "Was this project a success?"

4.1 Preliminary project requirements

4.1.1 Introduction

4.1.1.1 Purpose

This section is intended to present a detailed description of the high performance XML/XSLT transformation server being developed by the Oregon State University CS Capstone Team "XZES40". The intended audience for this section are the developers and sponsors of the project.

4.1.1.2 Scope

The name of this software, for lack of a better one, will be XZES40-Transformer.

The core product being delivered is a high performance XML/XSLT transformation server. This server will be able to perform repetitive document transformations quickly and efficiently by caching previously processed and compiled documents. Time is saved by pulling from an in-memory cache of documents and their compiled state rather than downloading and compiling documents which have already been processed, as current systems tend to do. XZES40-Transformer will also carry out transformations in parallel. It will transfer documents to and from clients via the HTTP or HTTPS protocol.

The target platform for XZES40-Transformer will be Debian Linux 8 ("Jessie"). The core product will be designed to allow the program to be ported easily from Linux to other operating systems like Windows and BSD.

In addition to the core server there will also be a command-line interface and web-interface developed to interact with the application, these will be called XZES-CLI and XZES-Web respectively.

4.1.1.3 Definitions, acronyms, and abbreviations

Below is a list of acronyms and abbreviations used throughout the document:

- Extensible Markup Language (XML): The human-readable data format used and processed by our application.
- Extensible Stylesheet Markup Language (XSLT): The human-readable format used to transform documents in our application.
- Xerces-C [**xerces**]: One library used to perform XML transformation in C.
- Xalan-C [**xalan**]: One library used to XML transformation in C.
- ICU [**icu**]: One library used to process UTF-8 character formated documents.
- Hypertext Transfer Procol (HTTP/HTTPS): The protocol over which XZES40-Transformer will interact with remote clients.
- HTTP Application Programming Interface (HTTP API): A standard way of communicating with a web application.
- Unified Resource Locator / Identifier (URL/URI): Addressable location of a resource over the internet (e.g., a website address).
- Debian 8 ("Jessie"): The target Linux-based operating system XZES40-Transformer will run on.
- Unicode Transmission Format 8 (UTF-8): The international standard for encoding text-based data.
- Apache Web-server: A Free and Open Source web-server.
- Common Gateway Interface Script (CGI Script): Server-side scripts that can run applications on client's behalf.

4.1.2 Overall description

The following sections of this document outline the factors that affect the creation of XZES40-Transformer at a high-level.

4.1.2.1 Product perspective

System interfaces

The XZES40-Transformer application will interface with the outside world over the internet via the HTTP networking protocol. The application will recieve HTTP POST requests to the application URI endpoint containing the documents to be transformed. Once the transformation is completed the transformed document will be sent to the user for download.

In addition to the transformed file the application will respond with an HTTP **OK** status. If an error occurs it will respond with a **SERVER ERROR** status and no file.

User interfaces

XZES40 will have two user interfaces which will access it's functionality over the internet.

- A Website to access XZES40-Transformer via a web-browser. This interface will be called "XZES40-Web".
- A CLI to access XZES40-Transformer via a terminal interface. This interface will be called "XZES40-CLI".

Both interfaces will not perform local document transformation. They will instead access the transformation service over the internet, making the HTTP API convenient to use.

Hardware interfaces

XZES40-Transformer will not have any direct physical interfaces as it is meant to be interacted with over HTTP or HTTPS. Any computer with an internet connection, monitor, input methods, and web-browser will be able to access XZES40-Transformer via the web interface. Any computer with an internet connection, monitor, input methods, and which has the CLI installed will be able to access XZES40-Transformer via the CLI interface.

The application will be targeted to run on a Debian Linux 8 ("Jessie") X86_64 CPU architecture server. This machine should have one port open for communicating over HTTP (port 80) and one for HTTPS (port 443) if that is configured.

Software interfaces

Below is a list of software required for the XZES-40 (on the host and on remote systems).

Name: Xerces-C++ XML Parser

Mnemonic: Xerces-C

Specification Number: XML 1.0 specification is implemented.

Version Number: 3.1.4 (Recent)

Source: http://xerces.apache.org/xerces-c/

Name: Xalan-C++ XSLT Processor

Mnemonic: Xalan-C

Specification Number: XML 1.0 specification is implemented.

Version Number: 1.10 (Recent)

Source: http://xalan.apache.org/xalan-c/

Name: International Components for Unicode

Mnemonic: ICU

Specification Number: Unicode 9.0

Version Number: ICU 58

Source: http://site.icu-project.org/download/58

Name: Apache CGI Processing

Mnemonic: Apache CGI

Version Number: Apache 2

Source: http://apache.org

Communications interfaces

An internet connection between the host and client is required for use of XZES40-Transformer. The host and client will be communicating over HTTP or HTTPS through the web interface or CLI interface. The application may be deployed behind a firewall.

Administrators may deploy multiple instance of the application, however additional instances will not be designed to communicate together, so they will each act autonomously.

Memory constraints

XZES40-Transformer will depend heavily on an internal caching system, so an appropriate amount of memory should be dedicated to the application. The specific amount of memory will depend on how much an instance of the application is expected to be used, however a minimum of 4GiB should be dedicated to the machine it is running on. That said, the more the merrier.

Operations

XZES40-Transformer will initially target the Debian Jessie operating system. To install the application a system administrator will acquire a Debian installation package (`xzes40-transformer.deb`), run the installation file, and begin the newly installed service.

The steps will roughly be as follows:

```
Download xzes40-transformer.deb
$ wget http://example.com/xzes40-transformer.deb

Install the package
$ dpkg -i xzes40-transformer.deb

Enable the xzes40-transformer Systemd service
$ systemctl enable xzes40-transformer

Start the Systemd service
$ systemctl start xzes40-transformer
```

Listing 1: Hypothetical installation setup steps

Installation on an additional Debian system would require the same procedure as listed above.

Installation on a non-Debian operating system will require a system-specific installation file, which we will create. The non-Debian systems we will include:

- Windows 7+
- MacOS
- FreeBSD
- RedHat Enterprise Linux

Once installed and setup, XZES40-Transformer will require minimal user-interaction by the system administrator. The application will run as a daemon on the host system. If a fatal error occurs the daemon will restart.

While users are not interacting with the application it will idle in the background. During periods of intense use the application will manage it's own resources to avoid breaking.

As the application does not store ephemeral data, there will not be a need for data backup nor data restoration.

Site adaptation requirements

XZES40-Transformer will use Apache to manage web-requests. If users want to setup secure communication over HTTPS they will need to do this manually using Apache.

The application will include a configuration file to specify resource limits, and other relevant information. This file should be tailored to a given user's installation and needs.

In addition to the configuration file, the user may configure the daemon through the daemon manager (`systemd` for instance) to set hard-limits on the application's resource usage.

4.1.2.2 Product functions

XZES40-Transformer will perform one function: XML/XSLT document transformation. Given one XML and one XSLT documents it will return a transformed XML document.

This functionality will be remotely accessible via an HTTP web-page and CLI interface.

4.1.2.3 User characteristics

The **user** of our application is expected to have common web-interface knowledge (e.g., they should know how to navigate a website, upload a file, and download a file).

4.1.2.4 Constraints

XZES40-Transformer will be subject to the following limitations:

- The code must be licensed under Apache 2.0.
- It must handle memory limitations gracefully.
- It must restart if a fatal error occurs.
- It must run on Debian 8.
- It must be accessible over a network.
- It must have an accessible interface.

4.1.2.5 Assumptions and dependencies

XZES40-Transformer will be written to interface with an OS agnostic API for any operating-system level operations (e.g., reading and writing from the cache). The application will be portable to new operating systems by writing an OS-specific interface layer and compiling the binary for the given target platform (e.g., Windows or FreeBSD).

XZES40-Transformer will assume that the relevant libraries and languages listed under Software Interfaces are already installed. The installation package we create will resolve these dependencies if they are not already installed with the correct version.

4.1.2.6 Apportioning of requirements

Development of the application, user interfaces, and installation packages will be carried out over a 19 weeks, split into three development cycles: Alpha, Beta, and Release.

The Gantt chart can be seen in the Appendix, Figure 7.

Alpha

During the Alpha phase of development we will collect benchmark data, create our basic transformation functionality, and begin work on Cache and Parallel computation optimizations. By the end of week three the application will be able to accept two input documents and output a transformed document. After the initial transformation functionality is complete work on optimizing this process will take place by adding caching and parallel processing to the transformation cycle.

Beta

During the beta phrase of development the XZES40 team will begin work on the HTTP API, the web interface, the Debian package, and further optimizations on the transformation process. Work on the web interface is not possible without the CGI interface first being added, however most other development can take place in parallel.

Release

During the Release phrase we will work exclusively on stretch goals including the CLI interface as well as the RedHat, BSD, and Windows packages. While these goals would be nice to achieve, we understand that there will probably be overflow from the Alpha and Beta phases of development, so we hope to complete all required deliverables well before the release deadline.

4.1.3 Specific requirements

4.1.3.1 External interfaces

XSLT40-Transformer will have two user interfaces: a **Web Interface** and a **Commandline Interface (CLI)**.

Web Interface

The website interface for XZES40-Transformer will include of a form with the following fields:

XML File A file upload field for the XML document.

XSLT File A file upload field for the XSLT document.

Output Filename (optional) The filename of the output document. If one is not specified a name will be generated of the following format `document-transform-<date>.xml>`.

The website will make an HTTP `POST` request to the server. This `POST` request will include an XML and XSLT document in it's payload.

The page will redirect the user to a new page where they can download the transformed file.

The Web Interface will require a web-browser (supporting HTTP4.0+).

A prototype of the web interface can be seen in the Appendix, Figure 2.

Commandline Interface (CLI)

The CLI will give users a text-based interface with XZES40-Transformer. The with the following flags:

`-xml-file=` Specifies the input XML document.

`-xslt-file=` Specifies the input XSLT document.

`-server=` Specifies which server to connect to (`e.g., http://servername.ext`)

`-output-file=` (optional) Specifies a file to write out to. Otherwise writes to a file of the following format `document-transform-`

`-port=` (optional) Specifies which port to connect through if non-standard (`e.g., 8001`)

`-help=` (optional) Prints out a help menu (describing these flags)

The CLI will take the following arguments and make a `POST` request to the server. The transformed file will be automatically downloaded to the user's desired location, or to the current working directory with the automatic file name.

The CLI requires a UNIX terminal and UNIX shell.

An example of some CLI interactions can be seen in the Appendix, Figure 3.

Both interfaces XZES40-Transformer will require a method of input (keyboard and mouse or touchscreen), an internet connection, and monitor.

As input both interfaces expect one XML 1.0 formatted document and one XSLT 1.0 formatted document. These files can be UTF-8 or ASCII character encoding.

As output both interfaces will send the user an XML 1.0 formatted document of UTF-8 character encoding.

4.1.3.2 Functions

The following functions will be the core functionality of XZES40-Transformer.

- `int transform(string XML_filename, string XSLT_filename, string output_filename)`: This function is called to transform the given XML_FILE with the XSLT_FILE. If output_file is defined the file will be written to that location. If output_file is not defined the new file will be written to STDOUT.
  Returns a status macro (SUCCESS or FAILURE).
- `type_cache* get_cache(input_filename)`: Returns a pointer to the cached file.
- `type_cache check_cache(string input_filename)`: The system will check if the given file is in the cache. Returns TRUE if the file is in the cache and FALSE if the file is not in the cache.
- `type_cache set_cache(string input_filename)`: The new XML file will be saved in the cache. Returns SUCCESS or FAILURe macro if the document was or was not successfully cached.
- `int compile(string input_filename)`:
  The system will compile the given XML/XSLT file into machine code to later be transformed.
- `int delete_cache()`: Removes old documents from the cache which are not being used. Triggered when the cache is filled to a certain capacity.

4.1.3.3 Performance requirements

XZES40-Transformer will perform better than existing Open Source XML transformation software. The application will have a higher rate transformations per minute, normalizing for input file-size. Given a standard set XML + XSLT document pairs, the application will complete the transformations on average faster than it's leading competitors.

The transformations will also be verified for correctness. It is expected that the application will perform document transformation with high correctness.

4.1.3.4 Logical database requirements

XZES40-Transformer application does not require database.

4.1.3.5 Design constraints

For the XML compilation and transformation process we are restricted to using the Xerces-C and Xalan-C libraries. For document encoding and decoding we will use the ICU UTF-8 libary.

Standards Compliance

Input documents must be correctly formatted XML and XSLT documents. Malformed documents will be rejected by the application.

Correctly formatted XML documents follow the W3C outlined XML 1.0 and XSLT 1.0 formats. [**xml-spec**] [**xslt-spec**]

Our application will also communicate with users over HTTP/HTTPS, however we are not implementing these standards, just using them to communicate over the internet.

4.1.3.6 Software System Attributes

Reliability

XZES40-Transformer will be reliable if it's cache is up to date, to avoid transforming documents incorrectly.

Availability

Since XZES40-Transformer will be run as a web-service it should be highly-available during business hours. It will be configured to handle heavy workloads and restart if it crashes. Upon restart it's cache will be cleared.

Security

XZES40-Transformer may be used to handle sensitive data, however it is not this team's job to account for that.

If administrators want the application to be secure they may choose to deploy it behind an organization firewall or configure the web-service to use HTTPS instead of HTTP. The application's HTTP traffic will be handled by Apache via a CGI script, this means that any Apache webserver configuration can be used with XZES40-Transformer's web API.

Maintainability

Our application will be deployed as a daemon which will be configured upon installation. This configuration may be modified after installation, but should work "out of the box".

Portability

As an Open Source project XZES40-Transformer will be designed for portability. It will perform all operating-system specific operations via an OS agnostic API. When the application is compiled on a new platform it will compile against the given OS API (e.g., Windows, Linux, MacOS, etc).

As for installation packages, one must be manually created for each platform, however this is not an urgent requirement and can be carried out platform-by-platform after the initial development is completed.

4.1.3.7 Organizing the specific requirements

System Features

Transformation

The core feature of XZES40-Transformer will be the XML + XSLT document transformation.

Stimulus:

As input the application will accept two files, one XML 1.0 formatted document and one XSLT 1.0 formatted document.

Response

As output the application will return one XML 1.0 formatted document. This will be the output of the program.

If a mal-formatted document is given as input the output of the program will be an appropriate error.

Caching

To increase performance the application will cache certain compiled documents.

Stimulus:

As input the in-memory cache will accept either a compiled file for caching or a hash of a compiled file for retrieval.

Response

If the cache is given a hash and "asked" for a cached document it will either respond with the contents of the compiled file or a FALSE response, signifying the document is not in the cache.

If the cache is given a file for storage it will respond with either a SUCCESS of FAILURE response if the document was successfully cached or not appropriately.

Parallel Computation

To further increase performance the application will carry out independent calculations in parallel.

This feature does not have easily defined "stimulus + response" pairs.

The application will need to be designed to avoid race-conditions during parallel computations.

Web API

The application will have a web-accessible API.

Stimulus

As input the API will accept either GET or POST requests. POST requests will contain a payload with the XML + XSLT document pair as listed above for the transformation feature.

Response

As response to a GET request the API will respond with a SUCCESS, saying that the application could be reached. As response to a POST request the application will respond with either a transformed XML document or an appropriate error.

User Interface

The application will have two user interfaces. These will allow the user to provide the above stimuli to the web API, download the response file, or display the error message from the API.

4.1.3.8 Additional comments

There are no additional comments to be made at this point.

5 Supporting Information

The following table outlines **required** project components ordered by importance:

Component	Owner
Document Transformer	Elijah C. Voigt
Website Interface	Shuai Peng
Web API	Elijah C. Voigt
Document Cache	Shuai Peng
Daemon Process	Elijah C. Voigt
Parallel Document Transformation	Shuai Peng
Benchmarking	Zixun Lu
Debian Software Package	Zixun Lu

Figure 1: Required project components ordered by importance

Component	Owner
CLI Interface	Elijah C. Voigt
CentOS Linux software package	Elijah C. Voigt
Windows software package	Shuai Peng
FreeBSD software package	Elijah C. Voigt

Figure 2: Stretch goal project components ordered by importance

5.1 Appendixes

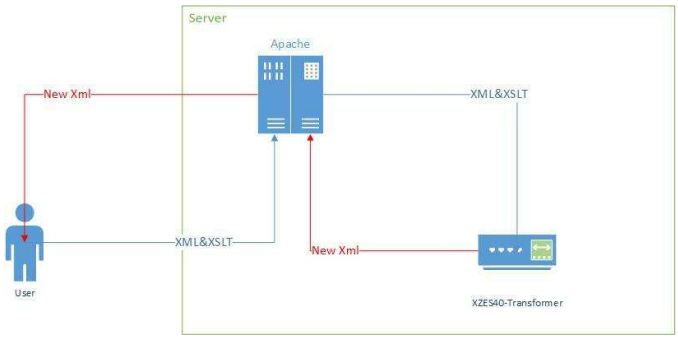


Figure 3: Diagram of dataflow. Documents are sent to the application through the Apache gateway interface. Once processed documents are returned to the user through the gateway interface again.

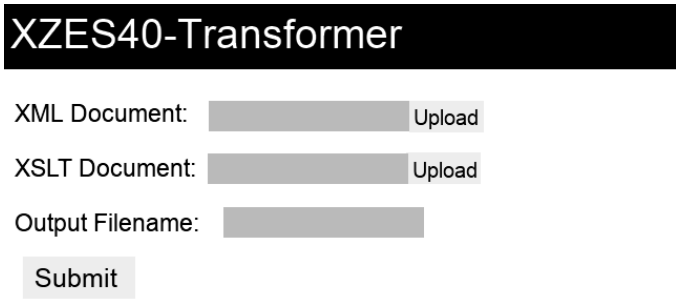


Figure 4: Prototype of the Web Interface. Demonstrates the simplicity of the interface and required form fields.

```
Normal usage:
$ xzes40cli --xml-file='./my-file.xml' --xslt-file='./my-other-file.xslt' --server='http://example.com/
    xzes40-transformer' --output-file='./newfile.xml' --port='8001'
Sending XML and XSLT files to http:/example.com:8001/xzes40-transformer
Transformation complete. Downloading response file to newfile.xml

Sending a bad file:
$ xzes40cli --xml-file='./badfile.jpg' --xslt-file='./badfile.txt' --server='http://example.com/xzes40-
    transformer'
Sending XML and XSLT files to http:/example.com:80/xzes40-transformer
ERROR: Server was unable to transform the requested files.

Using inadequate parameters
$ xzes40cli
Please provide an xml file (--xml-file), xslt file (--xslt-file) and a host (--server).
```

Figure 5: Example use-cases of CLI interface.

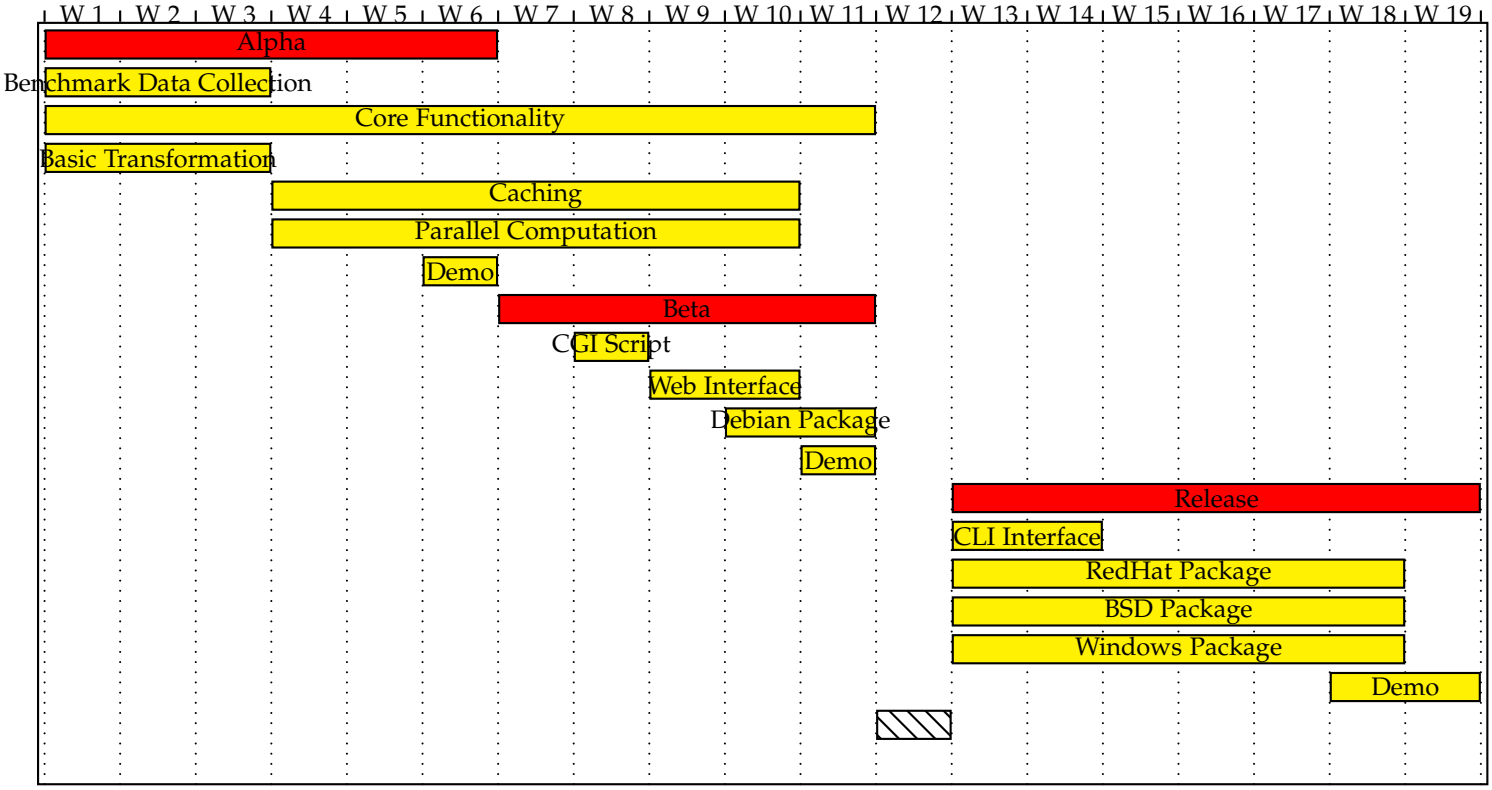


Figure 6: Development Gantt Chart Timeline.

5.2 Added requirements

This section outlines requirements added to the project during development.

#	Requirements	What happened	Comments
1	Parameter passing	This is a simple parameter passing requirement that is required by client in spring term. It just pass the parameter to our transformer, so transformer know what the output looks like	
2	Additional file passing	Some XML file and XSLT file require the DTD file which is a database file, so we add this function to our transformer, because we need benchmark our program.	

Table 1: Added requirements and rationales.

5.3 Updated requirements

This section outlines requirements which changed during development.

#	Requirements	What happened	Comments
1	Daemon	This requirement was more complex than initially expected, there was confusion about the complexity of this component. We were originally unclear about weather we needed to run a daemon for our in-memory cache, or if we could store the cache in a separate application like Redis.	
2	CLI Interface	A CLI Interface was not explicitly created for the application, however one was created for testing purposes.	With small changes this interface may be created if further development takes place

Table 2: Updated requirements and rationales.

5.4 Removed requirements

This section outlines requirements which were removed during development.

#	Requirements	What happened	Comments
1	Installation Package	This requirement was not explicitly removed from the requirements list, however no effort was made to complete this component of the project.	Neither the core requirements, a Debian package, nor the stretch goals, a Windows, CentOS, and BSD package, were pursued. The client agreed that as a prototype an installation package was not required to call the project as success.

Table 3: Removed requirements and rationales.

6 PROJECT DESIGN

6.1 Preliminary project design

6.1.1 Introduction

6.1.1.1 Purpose

The purpose of this document is to outline the entirety of the design of the XZES40 application for the purposes of referencing during development and for communication with the project sponsor(s).

6.1.1.2 Scope

The scope of this document is to outline in necessarily complex terms how the XZES40-Transformer application will be developed.

6.1.1.3 Development Time-line

The following Gantt chart outlines the projected time-line for development. This starts at the first week of January (winter-term week 1), and goes through project completion in June (end of spring term).

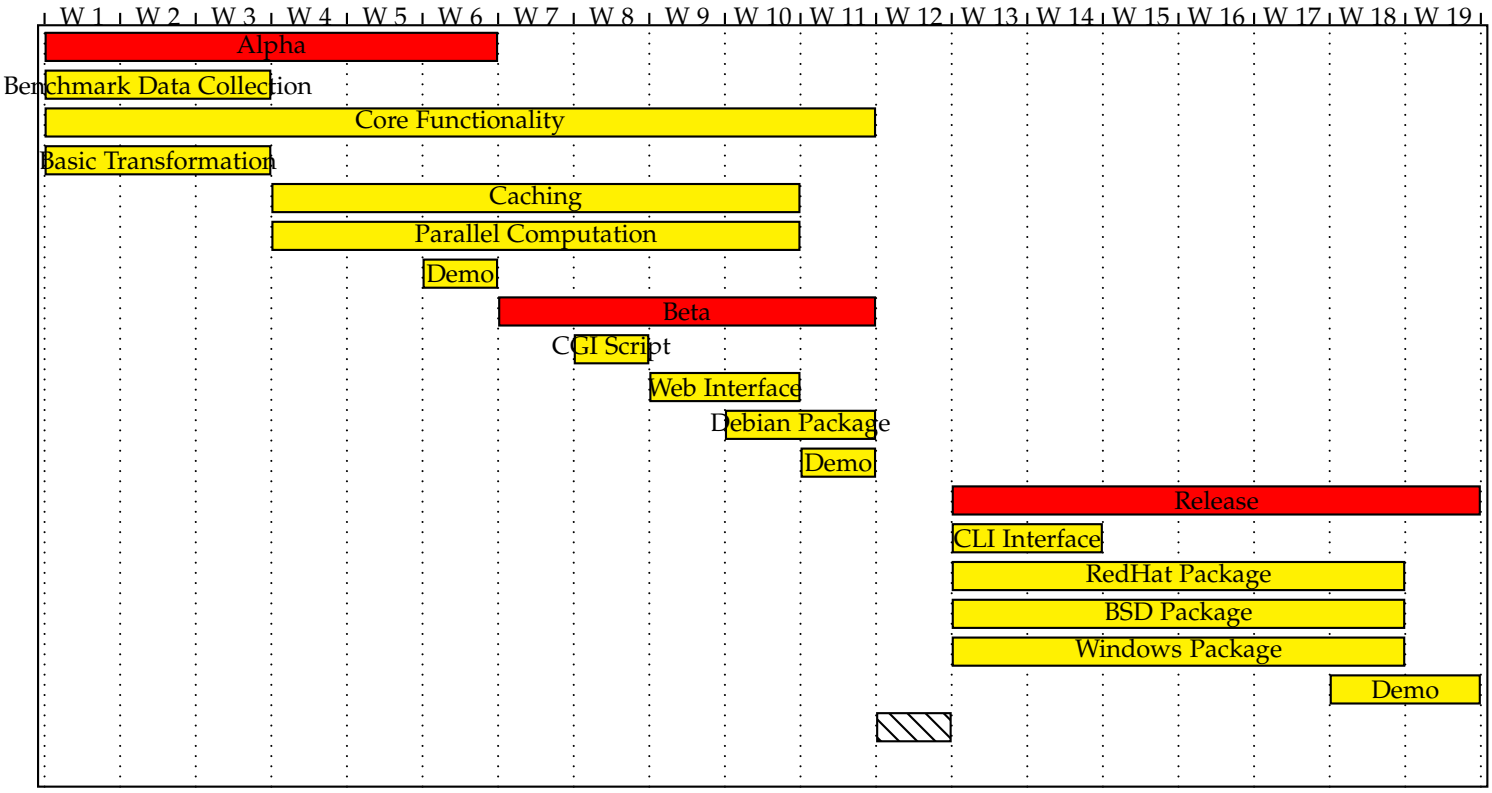


Figure 7: Development Gantt Chart Timeline.

6.1.1.4 Summary

XZES40-Transformer is an application which transforms one XML with one XSLT document for the purposes of data transformation, like that of a spreadsheet program like Excel. The uses for this rage widely from business to scientific to personal uses. For most needs, an XML/XSLT document transformer needs to be able to handle a high volume of requests by a wide variety of clients. To address these needs the XZES40 team is building an Open Source XML/XSLT document transformer with key optimizations built in to improve the document transformation process; helping businesses, institutions, and individuals get more done in a day. [**xml-spec**] [**xslt-spec**]

6.1.1.5 Issuing organization

This document has been issued by Oregon State University and the Apache Software Foundation through the OSU 2016-2017 CS Capstone class.

6.1.1.6 Change history

Revision	Date
Working Draft	2016-11-17

Figure 8: Change History Table

6.1.2 Component Overview

The following is a list of **required** project components ordered by importance:

Component	Owner
Document Transformer	Elijah C. Voigt
Website Interface	Shuai Peng
Web API	Elijah C. Voigt
Document Cache	Shuai Peng
Daemon Process	Elijah C. Voigt
Parallel Document Transformation	Shuai Peng
Benchmarking	Zixun Lu
Debian Software Package	Zixun Lu

Figure 9: Required project components ordered by importance

Component	Owner
CLI Interface	Elijah C. Voigt
CentOS Linux software package	Elijah C. Voigt
Windows software package	Shuai Peng
FreeBSD software package	Elijah C. Voigt

Figure 10: Stretch goal project components ordered by importance

6.1.3 Document Transformer

This subsubsection outlines the views and viewpoints related to the Document Transformer. These components relate to core functionality of the application:

1) Taking input XML and XSLT documents
2) Parsing them into DOM objects

3) Transforming those into a new XML document
4) Returning the final product to a user

This subsubsection does not necessarily outline how users interact with the application, for that one should see the User Interface subsubsection (**??**).

Transformer

As the name suggests, the Transformer is the core of the XZES40 Document Transformer, which is the core of our application as a whole. At a high level this component takes two documents, an XML and XSLT document, and returns a transformed XML document. The rest of the application is built around this component; all other parts of the application depend on or work toward this feature.

Context

The Transformer provides the core functionality of the application by transforming input documents into output documents.

All users of the application use the Transformer indirectly by way of using it to transform their input documents into output documents. The Transformer component is not be directly exposed, however it is accessible via the Web API (**??**).

Composition

The Transformer's functionality is outlined in the following steps:

1) Receive the input XML XSLT files.
2) Create a Document object out of the input files. This will either encode the input files as DOM objects (InputSource) or fetch the pre-compiled objects from the cache. (**?? ??**)
3) Pass the Document's parsed contents to the [**xalan-library**] **transform** method.
4) Encode the transformed document back to a text file and return this document.

These steps will each be held in a thread spawned by the long-running daemon process. (**??**)

Dependencies

The Transformer directly depends on the following internal application components for the following reasons:

- The Document class (**??**) is used to create a transformable DOM object from an input file. [**dom-spec**]
- The Document Cache (**??**) is used to store and retrieve parsed and transformed documents. While this cache isn't strictly necessary for document transformation, it is used to speed up the process drastically.

State Dynamics

The Transformer deals heavily with state dynamics in ways: transforming documents .

Transforming documents

The Transformer does not directly handle transforming documents, this task is delegated to the Xalan C++ library. [**xalan-library**] The transformer takes two or more documents as input: an XML DOM file and an XSLT object. The input documents are transformed by the Xalan C++ library and the resulting document is eventually returned by the writer (**??**) via the (**??**).

Interactions

Few components directly call the Transformer, however the transformer depends heavily on communicating with the cache. This is done by using a "cache" object. Documents are retrieved by using a "get" method and added to the cache with a "set" method. More information can be found in the Cache subsubsection of this document (**??**).

Interfaces

The Transformer has one programmer-facing interface, the "transform_documents(filepaths)" function. This can be used for testing, mocking, or implementation purposes.

transform_documents(filepaths)

- This function takes as argument the path to the XML file, the path to XSLT document, the output destination for the output file.
- As output it writes a transformed XML and returns a status code.

Document class

The Document class gets as input an XML or XSLT file prepares it to be transformed. It does this by either compiling it into a DOM objects and adding it to the cache or fetching the already compiled object from the cache.

Context

The Document class is a major component of the application. It receives as input an XML or a XSLT file, it then parses and store or just retrieves the DOM representation of that file. The user will not use this function directly, but it executed by the Transformer (**??**).

Dependencies

The Document class also depends on the Cache to store and retrieve parsed objects.

Interactions

1) The class obtains a Unicode formatted XML or XSLT file-path.
2) The class checks if the file has been parsed and stored in the Cache, and does not re-parse the file if it is in the Cache.
3) The class generates a DOM object via the Xerces C++ library.
4) The class stores the DOM object to the Cache.

Resources

The Xerces C++ library will be used during the parsing to generate a DOM object from the input file.

Interfaces

The following function declaration is used for the parser method:

class Document(source_file_path)

This constructor receives an XML file or a XSLT file, and then parses it into a DOM object which is stored in the Cache.

In parsing and storing the object, the class has methods which read the file contents, hashes the contents of the file, and uses this as a key when inserting data into the Cache.

Cache

The Cache is a plus one feature of the XZES40 application. The Cache speeds up document transformation by storing and retrieving previously parsed documents.

In practice the Cache will operate much the same as a Hash-map does, storing data at a location given a key which can also be used to retrieve the data. We will using keyList which is provided from XercessC.

Context

The program heavily depends on the Cache. The Cache can store, delete, and retrieve DOM object from the in-memory cache

Storing data

This is the major element of the Cache.

The Cache stores data from the user, primarily parsed DOM data to avoid re-compiling files.

Deleting data

This provides the ability to remove items from the Cache for whatever reason.

Retrieving data

This allows users to fetch information stored in the Cache given the object's key.

Composition

Below are few of the components making up the Cache.

- The Cache stores input data in a Struct along with the last time it was accessed and they key used to access the data.
- The Cache can retrieve the data via searching the associated key.
- The Cache can delete data corresponding with a key.

Logical

XZES40 handles object as a special structure. The Cache saves that object in the following format.

Key

This is a hash value associated with a cached object. This value is for retrieving the data from the Cache.

Content

The content is the parsed or transformed DOM object.

```
struct node {
  dom data;
  string key;
  date last_used;
}
```

Listing 2: Psuedocode for the caching generic object "node".

Information

If an object is not in the Cache, the Cache stores the object into the in-memory cache. Objects are removed from the cache via the Cache class' delete method.

State Dynamics

The Cache handles the following state changes in the following ways.

- When the Cache starts it allocates a block of memory for storage.
- If an item is being set the cache ignores previously existing data. It is the developers duty to ensure important data is not being overwritten.
- When data is being read the state of that item in cache is assumed not to change. If the item is not found it returns an empty object.
- When an item is deleted it returns a SUCCESS status if the object existed and is not deleted, and a FAILURE status if the object did not exist in the cache before the call.

Interactions

The parser sends a parsed DOM object to the Cache. The Cache will check the flies if it is exist in the memory. If the file exists in the memory, the Cache retrieves the DOM object from the Cache and return to the Document class. If the file does not exist in the memory, the Document class continues to process the file, and stores parsed file into the Cache.

Algorithms

The Cache can be divided into five major function. The First function is **set**, second is **get**, third is **dump**, fourth is **load**, and fifth is **delete**.

Set

Stores an object at a location in the allocated memory block based on the an MD5 hash of the "key" parameter given.

Get

This function retrieves the DOM object from the cache by the key of the parsed file.

1) If there is key exist in the cache, **return** the DOM object at that location.
2) If the key does not exist in the cache, **return** an empty storage struct object.

Delete

This function delete the DOM object from the cache by the key of the parsed file.

1) The Delete receives a key to delete the data it's located at.
2) If there is key exist in the cache, the Delete deletes this DOM object in the cache, and **return** that there is successfully delete.
3) If the keys is not key in the cache, **return** is no DOM object inside of cache.

Resources

The Cache needs a large memory allocation to function optimally.

Dependencies

The Cache depends on the Document and the Transformer to populate it. The Document class sends a file to the Cache for checking if file exists in the Cache. The Document class also can send a parsed DOM object for storing into the Cache if file does not exist in the Cache. The Cache may either return a DOM object to the Transformer, or the cache return null if the DOM object is not in the Cache.

Interfaces

Below are the major interfaces for the cache component of XZES40-Transformer.

int document_cache(dom_object file)

This checks the DOM object if it exist in the Cache.

- If it's not in the cache, return fail.
- If it is in the cache, return true.

dom_object set_cache(dom_object file)

This receives object file and store it into the memory.

dom_object delete_cache(dom_object file)

This deletes data from the Cache.

- If the DOM object exists in the cache, return true.
- If the DOM object does not exist in the Cache, return error.

dom_object get_cache(dom_object file)

This retrieves data from the cache.

- If the DOM object exists in the cache, return the DOM object.
- If the DOM object does not exist in the Cache, return error.

Parallel Computation

In addition to the Cache component of XZES40-Transformer (**??**), the application will also carry out certain computations in parallel to further leverage the computing resources available to it and compile documents even faster.

Context

The Parallel Computation component of the application carries out the following operations in parallel to speed up documentation transformations.

Document Parsing

This will be carried out in parallel. These operations are logically independent so they can be carried out simultaneously without affecting data integrity.

User Requests

This will also be carried out in Parallel, handled by Apache.

Dependencies

The Parallel Computation component of the application will be carried out at the high level by Apache delegating parsing jobs, see the Web-API subsubsection for more information **??**. the application will also carry out parallel computation internally (C++) using the MPI library.

Interactions

Our MPI-based thread computing will operate mostly autonomously, except when putting data into and fetching data from the application Cache (**??**). With the exception of interacting with the Cache each parsing thread will not interact with other internal components of the application.

Algorithms

One major concern with handling a cache by multiple threads and processes in parallel is avoiding data corruption. This is not a problem we have yet solved and further revisions of this document will elaborate on how we will handle this dilemma.

Interfaces

When documents are being parsed each parsed document will spawn it's own thread. This functionality is in the Document class component of the application in subsubsection **??**.

6.1.4 Daemon

Context

Because the application needs to be available continuously, and because it needs to store objects in an in-memory Cache, the application is daemonized. This means that it continues to run in a suspended state when it is not actively performing document transformations. When it does receive an incoming request it spawns a thread, does the transformation, and closes the thread. This allows multiple document transformations to happen in parallel conveniently.

Composition

The daemon is composed of two main components:

- A parent thread which will wait for a signal to handle a request. This signal is accompanied with some context for what task needs to be performed, i.e., a document transformation.

- A child thread is spawned for each incoming signal. The child process exits at the end of it's transformation after returning the transformed document or an error which is propagated up through the parent process to the signaling process.

Interfaces

The interface to the daemon is the same as that of the daemon. A CLI on the host running the daemon will accept an XML and XSL document as input, pass these to the daemon, and respond with an error or transformed document.

6.1.5 User Interface

The Document Transformer is fine and great, but without a user interface it's not useful. The following two subsubsections, the Web-API (**??**), Website interface (**??**), and CLI (**??**) together outline the ways users will will interact with the system described in the previous subsubsection.

Web-API

The Web-API is the standardized interface between the user interfaces and an instance of the application running on a host, communicating over HTTP.

Context

The users for the Web-API are those who use the application from the CLI (**??**) and (Website **??**). Both interfaces interact with the XZES40-Transformer host via standard HTTP request methods. But nobody should use the API directly as a UI is much easier than crafting an HTTP POST request.

Composition

The Web-APIis composed of the following components:

1) An Apache runs on the remote host.[**apache-server**]
2) The server manages a Python CGI script which handles accepting requests and sending responses.
3) The Python script calls the XZES40-Transformer application locally, passing input documents from a POST request and sending response files via the CGI interface.

Dependencies

The Web-API internally depends on an XZES40-Transformer binary which accepts an input XML file, and input XSLT file, and writes a transformed file to disk in a predictable location. The binary should also exit with predictable exit codes to communicate any errors or successes.

Interfaces

The Web-API communicates with remote clients via a standard HTTP API. Below are the requests a client can send and the possible responses:

POST /api/

This is a request sent to the API endpoint (`/api/`) with the intention of getting two input files transformed into a new document. This can respond in the following ways.

200 OK Means the transformation was successful. This response includes a body containing the transformed file and a URI to re-download the response file.

400 USER ERROR Means that the user sent malformed documents. This can include a document which does not follow the XML/XSLT standard to a document which does not have a readable character encoding. This response includes a body containing an appropriately specific error.

500 SERVER ERROR Means that the server experienced an internal error while processing the request. This includes fatal XZES40-Transformer errors. This response includes a body containing an appropriately specific error.

The POST request is a request with a Form containing input documents in fields titled XML and XSLT for an XML 1.0 formatted document and an XSLT 1.0 formatted document respectively.

GET /api/

This is a request sent to the API endpoint (/api/) with the intention of getting a status of the server. This can respond in the following ways:

200 OK Means the Web-API endpoint is active and functioning correctly.

404 NOT FOUND Means that the Web-API endpoint is not configured correctly or the user is accessing a page which is not available.

503 SERVICE UNAVAILABLE Means that the Web-API is setup correctly, but the application on the remote host is not operating correctly.

The GET request is an empty GET request to the servers "/api/" endpoint.

Python Interfaces

The following interfaces are used for implementing the above HTTP interfaces.

process_request(http-request req)

This method is used to receive an HTTP API request. It does this by reading the request header and deciding based on that to carry out one of the above responses. In the case of a POST request it uses an "exec" call to the local XZES40-Transformer application binary and responds with the error / output file of that application.

Resources

The Web-API has the following external dependencies:

- The **Apache** is used to process HTTP requests by running the Python CGI script.
- **Python 2.7+** is the programming language the Web-API is written in.
- **cgi** is a Python library for processing web requests in Python.
- **cgitb** is a Python library for developing with the cgi library.
- **mod_wsgi** is an Apache module for interacting with Python.
- **mod_python** is another Apache module for interacting with Python.

They should be installed via a system package manager or the Python package manager (whichever is appropriate) for the Web-API to operate correctly.

Website

The website interface is the major user interface for the XZES40 application. Website interface is a GUI interface for user.

Context

The Website interface is common interface for application. User can uses website to complete request with few of instructions. User can upload the XML/XSLT files that they want, and the website will give the feed back.

Composition

The web interface to access the application in a browser. Here is components of the Web-UI.

- An upload field for the **XML** file.
- An upload field for the **XSLT** file.

- A button for user send the request to server via **POST** method.
- After user sends a request to server the website will respond with a message to user. This message may be error or successfully upload or generate download link.

State Dynamics

There are few of states for website.

- If user upload bad files the website gives feedback that the files were bad.
- If user upload files and server transforming the documents the website gives the feedback that user upload files successfully.
- If user uploads files and server is down the website gives the warning that transformation service is not working.
- If user uploads good files and the server successfully transforms the documents the user is given a download link to the new file.

Interactions

The initialization status of website is waiting for user upload files. After user select flies that they want upload and click the submit button. Website will send the request to server via the Web-API. There are many possible feedback as following.

- If the files uploaded are bad the website pops up a warning about the malformed files.
- If the user uploads both an XML and XSLT file the user is alerted that the request is good to go and that the website will generate a download link for them.
- If the connection is broken the website will give them a warning that user should check the connection between client and server.

Interfaces

The Web interface is XZES40 main interface for user. The web interface asks user to upload XML files and XSLT files, and gives the feedback of result. Here is the diagram for website interface.

CLI

The CLI is used to interact with the system via a text-based terminal/shell interface.

Context

The users of this interface are individuals who either prefer the CLI over a web interface or for testing purposes as automating tasks with the CLI is very common.

Composition

The CLI, written in Python, and will be composed of the following components.

- The main function parses command-line arguments specified below in the “Interfaces” part of this subsubsection.
- A query is built for the server including the XMLand XSLT documents to the specified server.
- The query is sent to the server in a POST request.
- When the response is received either an error message is displayed to the user or a file is saved locally.

Resources

The CLI depends on the following system dependencies:

- Python 2.7+ is the programming language it will be implemented in, so a Python runtime will be necessary for the application to run.
- Requests is the standard Python library for interacting with servers over HTTP.

- A terminal and UNIX compliant shell will also be necessary to access the application via the CLI.

Interfaces

The CLI has the following options available at the command-line:

```
$ xzes40
    --server=<server-url[:port]> # API endpoint and port to be used.
    --xml=<input-file>.xml       # Input XML file
    --xslt=<style-sheet>.xslt    # Input XSLT file
```

Figure 11: CLI Flags. All fields encased in chevrons symbols are required. All fields with square brackets are optional.

Example Use-cases

The following are examples of the CLI in use:

```
$ xzes40 --server=http://xzes40.example.com:8080 \
         --xml=input-file.xml \
         --xslt=style.xslt
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Downloading transformed document to ./xzes40-transformer-2016-11-22.xml
```

Figure 12: “Happy path” use-case without specifying return file name.

```
$ xzes40 --server=http://xzes40.example.com:8080 \
         --xml=input-file.xml \
         --xslt=style.xslt \
         --output=transformed-doc.xml
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Downloading transformed document to ./transformed-doc.xml
```

Figure 13: “Happy path” use-case with return file name.

```
$ xzes40 --server=http://xzes40.example.com:8080 \
         --xml=input-file.xml \
         --xslt=style.xslt \
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Server responded with error. One of your documents is malformed.
```

Figure 14: “Un-happy path” with malformed document.

“Un-happy path” with bad host.

```
$ xzes40 --server=http://fakesite.com:8080 \
         --xml=input-file.xml --xslt=style.xslt \
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Unable to reach server. Is the host/port correct?
```

Figure 15: “Un-happy path” with bad host.

```
$ xzes40  --xml=input-file.xml--xslt=style.xslt
Please specify a host
```

Figure 16: “Un-happy path” with no host.

```
$ xzes40  --server=http://xzes40.example.com:8080
Please specify an input xml and/or xslt document.
```

Figure 17: “Un-happy path” with missing input document.

6.1.6 System Requirements

The XZES40-Transformer will be required to work on the Debian Linux system. Once development on Debian is completed the application will be ported to other OS platforms. Our team will release Debian, CentOS, BSD and Windows packages.

Installation Packages

For installation convenience the XZES40 project will provide a Linux, BSD, and Windows installation packages.

Context

The user can directly download installation packages through the internet and install the XZES40-Transformer on their local systems. We will create the Debian package initially. After this, we will create the CentOS, BSD, and Windows packages. We will upload those packages to the internet and the users can directly download the packages in their operation system.

Resources

The following tools will be used to create the installation packages.

Packages	Tools	Description
Linux & BSD Packages	• FPM	• Translates packages from one format to another • Allows re-use of other system's packages
Windows Packages	• WIX	• It is a open source project. • It is more stable and security than other tools. • It has steed

Figure 18: Installation Packages Resources

OS-API

The OS-API will be the programs interface with the OS. It will be created to make porting the application easier.

Logical

Any operating system specific operation will be wrapped by the API.

Interactions

The application will interact with the host system via an OS-API. This means that all operating system specific operations (e.g., read, write, seek, etc) will be done via an API. When the application is compiled on a new target platform (e.g., Linux, BSD, Windows) a new platform API must be created for compatibility.

Interfaces

The OS-API interface is for the developer. This interface performs operations by asking the OS-API to carry out the task and that request is translated to the system-specific system-call.

Performance Benchmark

Context

We will run test against similar application to determine how fast XZES40 should be. Throughout development we will put our application through the same paces and compare which is faster.

Resources

We will be bench-marking these applications specifically for the following reasons.

Technology	Description
Xalan C++ CLI	• Xalan C++ uses Xerces C++ to parse XML documents and XSLT. • The project provides an open source CLI program to test the project libraries. • Free and Open Source
Altova	• To meet industry demands for an ultra-fast processor. • It offers powerful, flexible options for developers including cml, python. • Superior error reporting capabilities include reporting of multiple errors, detailed error descriptions.

Figure 19: Bench-marking Resources

6.1.7 Design Rationale

This subsubsection explains why certain design decisions were made and "connects the pieces" of the application.

Cache Decisions

The reason the Cache uses an in-memory system rather than a database or a file-based cache is purely for performance reasons. Using an in-memory cache over a file-based one should yield faster performance. As a compromise the cache is dumped to a file periodically to save the state of the cache in case the application daemon (which holds the cache) is restarted.

This may cause problems as the service may run out of memory. In an attempt to mitigate this a Garbage Collector may be built which is triggered when a certain percentage of the application's allocated memory is used, or on certain time intervals.

Web-API Decisions

The Web-API connects the application to the outside world in an ideally simple to implement and quick to operate fashion. The application uses Apache to handle incoming requests, these are passed to a CGI script which could be written in anything, we chose Python because it is easy to write, maintain, and is well supported.

Python is called by Apache which then in turn, based on the request, either returns an application status or calls the XZES40 application locally. In this way Python with Apache is a simple, well supported set of tools which expose the application to the outside world.

These tools were not necessarily chosen for their speed, and so a redesign may be necessary if they create a bottleneck in the request pipeline.

Packaging Decisions

This document outlines the use of FPM to create its Linux and UNIX packages. This decision was made for convenience and to allow for quick iteration on the package. In doing research there *are* other tools which can be

used to create packages on CentOS, Debian, and BSD but they are very system specific and so would make iterating on the package very difficult, and updates to the software a pain to package. Using FPM we can even automate the build of packages on our UNIX-like systems.

Bench-marking decisions

To demonstrate the competitiveness of the XZES40-Transformer application we chose to compare the performance of this application against the Xalan C++ and the Altova application.

This feels like a fair and balanced comparison as we hope to be competitive with the Altova application but need to ensure we are at least better than the Xalan C++.

6.2 Changes

The design of the application changed in a few minor ways during the course of development. Some of these changes were a narrowing of the design while others were entirely added or removed components. This section outlines those changes and why they were made.

6.2.1 Daemon

The application daemon, the constant running process which processes incoming requests and returns a compiled document, was ill-planned early on in development. The ill-definition came mostly from a lack of understanding of how the caching process would work; it was undetermined if we would depend on a truly in-memory caching system or depend on some other system, like Redis, to store serialized files.

Once our client made clear that we should use an in-memory cache we were then tasked with determining the best way to execute this idea. We started by implementing UNIX sockets as a means of communication between our Python written CGI script and the C/C++ written daemon. This was shortly converted into a localhost network socket for convenience.

After we got communication between the Web API script and daemon we were then tasked with crafting an even loop and preventing race conditions. This touched every aspect of the C/C++ code, requiring a heavy re-write of the existing “main” daemon code as well as some components of the caching. The jist of this even loop is:

1) Setup networking connection with API script.
2) Enter while loop.
3) Enter for loop. Each loop here represents a thread being spawned.
   a) Receive a request. Format of request is specified in the next section.
   b) Lock cache.
   c) Process request.
   d) Unlock cache.
   e) Wait for another request.
4) After five requests have been processed, clean the threads and start the while loop again.

6.2.2 Web API

The web API is used to access the application over an HTTP(S) connection. It is processed as an Apache CGI script written in Python.

When a request comes it it is formatted as an HTTP Form with the following fields and values:

- xml: Content of XML input file.
- xsl: Content of XSLT input file.
- params: A JSON formatted key:value pair of custom parameters (discussed further in the next subsection).

This is processed using the CGI script library. The input files are hashed, the resulting hash is each file's name; the files are saved on disk to "/tmp/xzes/...". The two files and their parameters are concatenated and hashed together, this is the job ID and filename of the output file. A quick check is made to see if this combination of files and parameters has already been processed. If it has, the previously processed (and already locally saved) file is returned. If the output file does not already exist, the job continues.

The job request is formatted as "job_id, /path/to/input.xml, /path/to/input.xslt/path/to/output.xml, param1, 'val1', param2, 'val2', ...'. The serialized request is sent over a local network to the C++ daemon and processed. Since both the API script and daemon are on the same machine the files are opened directly from disk and the output file is written to the same directory.

When the request is over the daemon responds with a string "job_id,/path/to/output.xml,errors if any". The connection is then closed and the API send the file back to the user that made the request.

6.2.3 Custom parameter and file passing

One *very* late addition to the application was custom parameter and additional file passing to the application. The former is when a user sends a key:value pair to the transformer, each instance of the XML variable "key" is replaced with "value", usually being replaced with pre-defined default. The latter is when a user sends an additional(s) file to the transformer which are required for the transformation to take place; these can be thought of as header files.

The parameter passing was defined in the above section. Essentially the parameters are parsed by the web form (discussed below), serialized in JSON, and sent to the API to be decoded in python.

The file passing was not designed as it was never feasible to add this feature to the application.

6.2.4 Interface (web)

The web interface is written in stock HTML, CSS, JavaScript, and jQuery.

- The page is a single form consisting of two fields, one to upload the XML file and the other to upload the XSL file, an "add parameter" button, and a "submit form" button.
- That buttons adds a field for the key:value pair passed to the XML transformer.
- When the form is "submitted" jQuery processes the form, serializes the parameters into JSON, and sends it asynchronously to the API endpoint "/cgi-bin/".
- If the transformation is a success, a button prompting the user to download the file and another to preview the file in the browser appear in the page.
- If the transformation is not a success an error message passed from the transformer is displayed instead.

6.2.5 Interface (CLI)

The CLI interface was never developed in favor of the web interface.

As it turns out the web interface was only marginally more difficult to develop than the CLI.

A CLI used for testing does exist in the source repository.

6.2.6 Installation packages

In addition to the CLI, the administrator installation package was never pursued.

Bash scripts were used to provision a development VM, these may be used to *create* a development package, but one was not created for either Debian nor the other [stretch goal] targets.

This requirement was not explicitly dropped, just not pursued because of development constraints.

6.2.7 OS Agnostic C/C++ API

As development efforts were stretched thin, the dream of writing an OS agnostic API for system calls was not pursued as well.

This requirement was not explicitly dropped, just not pursued because of development constraints.

7 TECHNOLOGY REVIEW

In creating a piece of software it is best to look around, see what exists, and make an informed decisions about the tools you should use to carry out the project. This section outlines those technologies investigated and the reason the tools chosen were used.

7.1 Preliminary technology review

This is the preliminary technology review with minimal edits. It includes the original references page for authenticity.

7.1.1 Introduction

The following tables illustrates who is responsible for each component of the application.

Required functionality:

Section	Author
Research and Benchmarking	Zixun Lu
XML/XSLT Document Transformation	Shuai Peng
XML/XSLT Document Parsing	Elijah C. Voigt
XML/XSLT Document Caching	Shuai Peng
XML/XSLT Document Parallel computation	Zixun Lu
Web API	Elijah C. Voigt
Web Interface	Shuai Peng

Table 4: Responsibility chart for each core component of the project.

Stretch goal functionality:

Linux Package	Zixun Lu
Command Line Interface	Elijah C. Voigt
Windows Packae	Shuai Peng
BSD Package	Elijah C. Voigt

Table 5: Responsibiity chart for each project stretch-goal.

CONTENTS

7.1.2 Research and Benchmarking

7.1.2.1 Options

7.1.2.2 Goals for use in design

Our team will do the research and benchmarking to grantee that our application is fast enough.

7.1.2.3 Criteria being evaluated

We will test some number of requests against a comparable to find which transformers use the time less. Throughout development we will put our application through the same paces and compare which is faster.

7.1.2.4 Comparison breakdown

Technology	Description
Xalan CLI [**Xalan-C**]	• Xalan-C++ uses Xerces-C++ to parse XML documents and XSL stylesheets. • The project provides an open source CLI program to test the project libraries. • Free and Open Source • It works on the Debian operation system.
Altova [**Altova**]	• To meet industry demands for an ultra-fast processor. • It offers powerful, flexible options for developers including cml, python. • Superior error reporting capabilities include reporting of multiple errors, detailed error descriptions. • It only works in the Windows operation system.

Table 6: Technology evaluated for benchmarking our application against competing software.

7.1.2.5 Discussion

Xerces is a simple CLI application developed by the Xerces project to test the library. This is very similar to our program as it is open source, uses the same libraries, but lacks the caching we will implement.

RaptorXML is built from the ground up to be optimized for the latest standard and parallel computing environments. It is proprietary tool which we may try to out-perform as a stretch goal, but to start with out application will not try to out-perform.

7.1.2.6 Selection

We will compare our application to the Xalan-C CLI as it is the closest competitor to our application.

7.1.3 XML/XSLT Document Parsing (Elijah C. Voigt)

XML document compilation is the process by which an XML formatted document is taken and parsed into an in-memory object. The library we will be using, as requested by or client, will be the Xerces-C/C++ XML parser library; this review will evaluate that library. The other point of wiggle-room we have is in what way we parse and store the document in memory.

7.1.3.1 Options

For this requirement we will review the C/C++ library that we were requested to use by our client, and two XML parsing techniques we may use. The two techniques for parsing an XML document include SAX and DOM, the library

we need to use is able to perform both methods. These are both designed with specific use-cases in mind, however the method we ought to use is not so simply chosen.

7.1.3.2 Goals for use in design

With the goal of optimizing peformance in mind we will choose the parsing option which is fastest for our application. This is not as simple as choosing the one which is known for being fastest, because we will also be caching documents and performing parallel computations which may pull the needle toward one method of compiliation over another.

7.1.3.3 Criteria being evaluated

We will be evaluating the parsing method which is likely to perform best in our application given that we will be caching documents and compiling them in parallel.

7.1.3.4 Comparison breakdown

Technology	Description
Xerces-C/C++ [**xerces**]	• Requested to be used by our client. • Implements DOM parsing. • Implements SAX parsing. • Feature Rich.
DOM parsing [**dom-vs-sax**]	• Memory intensive. • Ideal for carrying out many operations on a document.
SAX parsing [**dom-vs-sax**]	• Memory light. • Event-based processing. • Ideal for minimal transformations.

Table 7: Technology evaluated for XML document parsing.

7.1.3.5 Discussion

DOM parsing is know for being memory intensive and slower in simple cases. It is also know for being ideal in applications like updating a web-page where many operations happen to a document. It is not necessarily ideal for simple document transformation, but does produce a complete parsed object which we can cache easily.

SAX parsing is known for being better than DOM parsing in that it parses a document in an event based method, giving it a smaller memory footprint and shorter time-to-transform. Simple transformations are common use-cases for SAX parsing. SAX does not produce an object model and instead uses callbacks to perform document transformations.

7.1.3.6 Selection

We will be using the Xerces-C/C++ library to accomplish the task of XML document parsing. It is feature rich enough to give us leeway in our development where we need it The library is also lean enough that it should not affect performance negatively.

As for choosing between SAX parsing vs DOM parsing, we will most likely choose DOM parsing since it fits our application's caching requirement well. DOM parsing produces an easily cached tree object which we can store and

retrieve for later operations. SAX parsing is faster than DOM parsing, but the time saved by using a cached object instead of re-parsing an XML/XSLT document dependency will likely outweigh the benefits of SAX parsing. It may be worth-while to investigate using SAX parsing early on in development if we find a notable performance boost, so some amount of SAX proof of concept work should be done for the sake of being thorough, but DOM parsing will be the targeted document parsing method.

7.1.4 XML/XSLT Document Transformation

7.1.4.1 Option

There are three options for the XML/XSLT Document Transformation. The first option is Xalan-C++, and the second option is Saxon C, the third option is Sablotron.

7.1.4.2 Goals for use in design

The major function of XZES40-Transformer is XML/XSLT document transforming. Apache foundation provide many ways to achieve this function.

7.1.4.3 Criteria being evaluated

The XML/XSLT document transformation is our major function for XZES40-Transformer application. We want this transformation compatible with good parse tools, and version of XSLT and XPath.

7.1.4.4 Comparison breakdown

The first option is Xalan-C++, this technology is required by our client, and this technology is supported by Apache. Xalan-C++ is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. Xalan-C++ contain the XercesC tools, so we don't need consider the parse tools.

The second option is Saxon C, which is Saxon company project. It performs the same operations as Xalan-C++, but it supports different version of XSLT and Xpath.

The third option technology is Sablotron. It used same version of XSLT and Xpath with Xalan-C++, but it designed to be as small program.

Technology	Description
Xalan-C++ [**xalan**]	• Xalan-C++ is open source project developed by Apache. It is implemented by XSLT version 1.0 and XPath version 1.0. • Xalan-C++ uses Xerces-C++ to parse XML documents and XSL style sheets.
Saxon C [**Saxon_c**]	• Saxon C is open source project. It is implemented by XSLT 2.0/3.0 version and XPath version 2.0/3.0. • Saxon C use different parse tools to handle the date.
Sablotron [**Sablotron_intro**]	• Sablotron is open source project by gingerall. It is implemented by XSLT 1.0 and Xpath 1.0. • Sablotron need extra parse tools to complete transformation.

Table 8: Technology being evaluated for XML document transformation

7.1.4.5 Discussion

The Xalan-C++ is the most powerful and popular XML/XSLT document transformation library and is used in many propetary as well as open source tools. It a robust implementation of the W3C recommendations for XSL Transformations (XSLT) and the XML Path Language (XPath). The Xalan-C++ is continuing update, and it is good tools for transforming documents. However, Xalan-C++ has poor structure API reference, and it makes developer hard to read and understand. Thus, developer need spend time on doing research with API.

The Saxon C is a good alternative to Xalan-C++. Saxon C can handle higher version of XSLT and Xpath, however implements different parse tools, so we need find out other technology.

The Sablotron does the same work as the Xalan-C++, but Sablotron is designed to be as compact and portable as possible. The size of Sablotron is much small than Xalan-C. However, Sablotron is old and has not been updated it since 2006.

7.1.4.6 The Best option

We will choose Xalan-C++ as our solution technology. The first reason is that Xalan-C is required by our client. We want to using Xerces-C as our parse tools, so Xalan-C is the best choice. Xalan-C++ is easy to use with clear example.

The second reason is that Sablotron is not stable. Even they are open source project, but the community has forgetten it. Saxon C is a good second option if client require different tools to complete XML/XSLT document transformation.

7.1.5 XML/XSLT Document Caching

7.1.5.1 Option

There are three technology options for cache. The first option is storing cache into memory. The second option is storing our cache in binary file on disk. The third option is to create database to handle all of data.

7.1.5.2 Goals for use in design

Caching is the "plus one" function of XZES40-Transformer application. Other similar application compile files each time and this wastes a lot of time and resource. We will create cache to solve this problem.

7.1.5.3 Criteria being evaluated

We want save the time and resources in our XML transformer, so efficiency is the most element that we consider. This is not only the speed of reading and writing from the cache, we must also weigh the persistence of the cache to avoid recompiling when the system (application or host) is restarted.

7.1.5.4 Comparison breakdown

The first option is storing the cache in-memory, this is the faster and easy way to store cache.

The second option is to create a binary file with the cache. When we are run our application, we read the cache data in from the cache file into memory. When the cache is updated it is written back to the original file.

The third option technology is that we create database to handle memory. This spends time to design and create database.

The fourth option is to use the in-memory key-value store Redis [**redis**] which is a popular for this kind of task.

Technology	Description
Memory	• Application check data from memory, and put cache into memory. • Retrieving data from memory is the faster way.
Temporary binary file	• Application loads binary file when it starts. After we close it, application save binary file in external storage driver. • Loading temporary binary file spend time, so it is slower than memory.
Database	• Application access data from database. • It takes time to create and manage a database.
Redis	• Objects to be serialized before being cached. • Adding an additional dependency to the project. • Less development time spent daemonizing and designing in-memory caching system.

Table 9: Technology evaluated for document caching.

7.1.5.5 Discussion

Storing cache into memory is the most easy way, we just need to allocate memory. However, the main drawback of this technology is when we close application, all of cache data will be wiped out. We have to compile file next time when we start running the application. When we are developing the XZES40-Transformer, we find a tools that KeyList is built in XercesC. This tools is helpful for managing memory, and it has good data structure.

Creating a binary file can avoid losing cache data, but it spends time to load file into memory when application starts.

Creating database is bad option for XZES40-Transformer application because access database spend resource, and it waste time to search cache data.

Redis is an appealing idea, but because it is an additional dependency, and serializing our parsed documents is not something our client wants our application to do, we are not able to pursue it. If our application had slightly different requirements this may be a viable option.

7.1.5.6 Selection

The best option technology is that storing cache into memory. Although it will lost data after close application, it save the time, and it the faster way. We may add a 'backup cache' solution to make this the best of both worlds, restoring from the backup when the system restarts but working mostly in-memory. And we will using KeyList, because it is built in XercesC. We can just add API, and easy to control the cache storing.

7.1.6 Parallel Document Transformation

7.1.6.1 Options

7.1.6.2 Goals for use in design

XZES40-Transformer will be increased performance by using parrallel computation. The parallel processing of the containment queries against an XML document utilizes parallel variants of the serial algorithm. There are two ways to do the prallel computation.

7.1.6.3 Criteria being evaluated

XZES40-Transformer use the parallel computation method do the same processing and operating in parallel. The criteria for a good tool in this area is mostly how easy to write code it is with the tool and how maintainable the code is, and of course how fast the eventual program is.

7.1.6.4 Comparison breakdown

Technology	Description
POSIX Threads [**posix-threads**]	• Defines a set of C types, functions and constants • Spawn concurrent units of processing • Achieve big speedups, as all cores of CPU are used at the same time.
OpenMP [**Openmp**]	• An API that implements a multi-thread, shared memory form of parallelism. • Uses a set of compiler directives • Take care of many of low-level details
MPI [**mpi**]	• Core syntax and semantics of library • Complexity, scope and control • Manage allocation, communication, and synchronization of a set of processes

Table 10: Technology evaluated to perform parallel computation of XML transformations

7.1.6.5 Discussion

Pthreads is a standard for prgramming with threads, it can achieve big speedups, as all cores of your CPU are used at the same time. OpenMP uses a set of compiler directives that are incorporated at compile-time to generate a multi-threaded version of your code. MPI allows us to manage allocation, communication, and synchronization of a set of process.

7.1.6.6 Selection

We will use MPI because it is a high-level standard, and so it will be easy to develop with.

7.1.7 Web API (Elijah C. Voigt)

7.1.7.1 Options

Our web API may be implemented via a C++ web application, a Python or Ruby web application, or an Apache webserver CGI script. Each of these has pros and cons, and each get the job done at some cost and with some benefits.

7.1.7.2 Goals for use in design

XZES40-Transformer will be accessible via a web API. This can be implemented a few different ways, but all of them must accomplish the same goal of allowing people to use the service over a network. The three options being evaluated here vary in how they achieve this goal, and so they represent more their technology and less the specific implementation which will be used.

7.1.7.3 Criteria being evaluated

The core of this application is related to document transformation. The more time that is spent on non-document transformation tasks should be kept to a minimum. Any technology we use to implement our web API should be simple, easy to develop, and easy to maintain. In short, *keep it simple stupid*.

7.1.7.4 Comparison breakdown

Our first option is to use an Apache CGI script, which would be a simple Python, Perl, or Ruby file which calls our C program and returns the results (transformed document or error) to the user, all via an Apache server gateway. The second options is to write a native web application using a web-app framework like Kore to handle HTTP requests. The third option is to use a python framework like Flask to handle HTTP requests. Each of these would be something that calls our document transformation app and exposes it to the outside world. How we handle that is important to consider.

Technology	Description
Apache CGI Script [**cgi-tutorial**]	• HTTP requests are handled by the Apache web-server. • XZES functionality is called by "shelling out" to the program and returning the results. • Apache CGI Script requires a running Apache Server on the host.
Kore web-app framework [**kore-io**] [**kore-feature**]	• HTTP Requests are handled by the Kore framework. • XZES functionality is called natively with C code. • Kore web-app framework acts as an independent daemon.
Flask web-app framework [**flask-site**]	• HTTP Requests are handled by the Flask framework. • XZES functionality is called either natively or by using `exec` to "shell out". • Flask web-app framework can act as an independent daemon or be managed by Apache.

Table 11: Technology being evaluated for the implementation of the web API.

7.1.7.5 Discussion

The above three technologies are all entirely valid choices for our application; each approach the problem from different angles.

The Apache CGI choice is the Occam's Razor solution relative to the others. Using the Apache web server we can register a script (written in Python, Perl, Ruby, etc) to accept requests and return a response. This is simple, maintainable, and easy to create; this is especially appealing if we are only concerned with implementing the API and not fancy features like accessing a database or storing user sessions. This option also allows us to leverage existing Apache Web-server

power like load balancing and simple authentication without needing to write those features ourselves, they're just an Apache configuration file away from being a reality.

Kore is a web-app framework which would allow us to develop our application in C/C++, which has it's pros and cons. C/C++ is notoriously difficult to write, and harder to write *well*, so it may be a time-sink. That said, it is nice to have a project which is written entirely in one language as contributors (ourselves and others) do not need to learn multiple languages to contribute to the transformer project.

Flask is another web-app framework, but one which is substantially easier to read and write. This has the benefit of being easier to maintain than a Kore framework, and we can write exactly the level of complexity we want from our API. On the other hand we would need to maintain a knowledge base of python, python frameworks, and python dependencies, so this is versatile but ultimately not necessarily easier to maintain than a kore framework.

7.1.7.6 Selection

Since we are working with Apache on this project, we want to develop a simple solution to our API problem, and the Apache Web server is a powerful tool we will choose to use this in our design. We will write a simple CGI script (which calls our C binary) and hook this into an Apache Web server. We will need to depend on the Apache Web server for our project's package, but this should not be as hard as writing a web app ourselves. We can also use the server to easily host our Web UI, which is a nice bonus.

7.1.8 Command Line Interface (Elijah C. Voigt)

7.1.8.1 Options

For the CLI we can develop a native C client, a simple Bash client, or split the difference with a Python client.

7.1.8.2 Goals for use in design

Each of these CLI tools must be able to construct an HTTP POST request with two documents, send that request to a server, and parse a response payload.

7.1.8.3 Criteria being evaluated

The criteria for this, as it is not part of the core functionality, is simplicity to write, maintain, and use for end-users.

7.1.8.4 Comparison breakdown

Technology	Description
C/C++ []	• Keeps the code base exclusively C/C++. • Can be distributed with system package managers. • Difficult to write and maintain.
Python []	• Simple to write and maintain. • Requires few external dependencies. • Can be distributed with the `pip` package manager.
Bash []	• Very simple to write. • Quick to write using existing system tools like `curl` or `wget`.

Table 12: Technology evaluated for the command-line interface.

7.1.8.5 Discussion

The C/C++ option is not necessary at all. This is an option which ought to be considered, but ultimately isn't worth pursuing as it will be very complicated to write and maintain, especially when simpler script-based options exist.

Python is a great middle ground. The language comes with many web-request libraries and provides tools for users to upload their application to the `pip` package manager. This means we can write a tool which performs well, is easy to maintain, requires few external dependencies, and can be downloaded by a package manager. It will also be available on all platforms which run python, which includes Unix and Windows systems.

Bash is a viable candidate, especially when other tools like `curl` and `wget` will make quick work of the task. The downside to this is that we cannot host our CLI on any standard package manager for verifiable distribution. This forces users to download the CLI from our website directly. Ultimately this will probably be used as a proof of concept, but it is not a final product.

7.1.8.6 Selection

We will start by creating a CLI in Bash for testing purposes, and if time allows we will create a more polished CLI in Python. Python language offers the best of both worlds in terms of simplicity, maintainability, and ease of use for end users, but for the sake of "getting something out the door" we will first create a tool that works in Bash.

7.1.9 Website UI

The Web Interface will be a simple webpage which calls our web API. For this reason this section focuses mostly on design technologies and less on HTTP request-handling technologies.

7.1.9.1 Option

This document reviews three possible technologies we can use to implement our website . First is plain-text HTML, CSS and Javascript, second is the Bootstrap front-end framework, and third is the Foundation framework.

7.1.9.2 Goals for use in design

XZES40-Transformer will have a web-based user interface, which will also be our main user interface. In considering which technology to use we will focus on making it conform with modern website design practices. This will allow us to write an application which is hopefully user friendly and intuitive to use. The website should also be look good.

7.1.9.3 Criteria being evaluated

The most important aspect to consider for this user interface is the appearance. We want to create web pages that work well on most screen sizes. So the cost, efficiency, visual appeal, and dynamic screen adjustment are what we consider most important in our technology of choice.

7.1.9.4 Comparison breakdown

- Cost: All of our options are open source and totally free and offer free documentation / tutorials.
- Efficiency: Using plain CSS/Javascript/HTML will perform well on most end-user's web-browsers, however it will be difficult to optimize the website to be responsive and adjust for smaller screen sizes. Bootstrap and Foundation were created with responsive design in mind. Both of them can change the size automatically in different size of screen. Both of them provide templates for creating web pages, but plain CSS do not provide.
- Learning speed: CSS is the basis of HTML style sheet. It is easy to understand and learn, however it hard to make it look good. Bootstrap and foundation spends time to learn, but after we learn the basically knowledge, bootstrap and foundation will be faster than CSS.

Technology	Description
CSS [**CSS_intro**]	• CSS is open source, and it the basis style sheet for HTML. • CSS is not able to easily create a web page that fit in different size of screen. • CSS is easy to learn, but it hard to use to make a good website.
Bootstrap [**boot_intro**]	• Bootstrap is open source project with good forum support. • Bootstrap is efficient because it has responsive deign. It provide many templates. • Bootstrap is easy to learn and use.
Foundation [**foundation_intro**]	• Foundation is open source project. • Foundation is efficiency, and it has responsive deign. It provide many templates. • Foundation is easy to lean and provides free tutorials.

Table 13: Web frameworks considered for the web-based frontend of XZES40-Transformer

7.1.9.5 Discussion

The table show us that all of them did similar work however, they have different advantages and drawback. CSS is basis of HTML style sheet, and it works great, but it hard to create web pages beautiful using *just* CSS and HTML. This make it hard to move web page into different size of screen, so we don't want to take CSS as our solution technology.

Bootstrap and foundation do similar work, and both are open source and free to use. However bootstrap is much more stable and provides more templates, and there is free instructions in the W3C school for using it.

7.1.9.6 Selection

The best option is the Bootstrap for our project because Bootstrap is open source project and has good tutorials in W3C schools. Bootstrap is the most popular font-end framework for web design, and it still updated by thousands of people. During we are developing our code, I think that we should have second choice. The Second choice is plain-HTML interface. Althoght plain-HTML dose not look beatiful, it is most straight way to present information to user. Keeping simple is the best way.

7.1.10 Debian & Centos Package

7.1.10.1 Options

7.1.10.2 Goals for use in design

Our team will release Debian and Centos packages. Users can directly download these from the website and use these packages to install XZES40-Transformer on their Debian and Centos operation system.

7.1.10.3 Criteria being evaluated

We may use Centos and Debian tools to build our package. We can use Centos operation system to build Centos packages and use Debian operation system to build Debian packages. Those tools are free to use.

7.1.10.4 Comparison breakdown

Technology	Description
Centos packaging tools [**centos-tool**]	• Use in Centos operate system • It is esay to use • Free.
FPM [**fpm-home**]	• Translates packages from one format to another. • Allows re-use of other system's packages. • Free.
Debian packaging tools [**debian-tool**]	• Use in Debian operate system • It is easy to use • Free.

Table 14: Technology evaluated for the Linux system package.

7.1.10.5 Discussion

The above tools are all valuable. If we were going to just develop a Debian package we may only use the Debian tools, and the same goes for CentOS. These tools are good at creating packages for those specific platforms, but since we intend to develop tools for mutiple platforms (Linux and BSD) using FPM to create cross-OS packages would be very convenient.

7.1.10.6 Selection

In the end we will use FPM to develop our packages becuase it makes life very convenient.

7.1.11 BSD Package (Elijah C. Voigt)

7.1.11.1 Options

XZES40 Transformer will be created with an installation package so that it can easily be installed on a host system. One platform we will create a package is FreeBSD. This package will be a stretch goal. This can be created manually, with FreeBSD system tools, or with a more general system-package creation tool.

7.1.11.2 Goals for use in design

In an ideal world this tool would be easy to use, and would automatically take our source code, compile it, and package it for distribution. Unfortunately that isn't a feasible solution for such a small project, but in our design we will be considering anything that gets us close to that reality.

7.1.11.3 Criteria being evaluated

The technology chosen should be easy to use, and allow for modifications to the host package in an expedient way. Creating system packages can be a pain, but creating one *incorrectly* is even worse. The tool chosen should be free, relatively painless to use, and require minimal human interaction to prevent the introduction of human errors.

7.1.11.4 Comparison breakdown

Technology	Description
Poudriere [**poudriere-tutorial**]	• Creates builds for multiple platforms including other versions of FreeBSD and other CPU architectures. • Builds packages in parallel. • Free.
FPM [**fpm-home**]	• Translates packages from one format to another. • Allows re-use of other system's packages. • Free.
pkg-create [**pkg-create-man**]	• Simple to use. • Installed on most FreeBSD systems. • Free.

Table 15: Tools considered for the use of system package creation.

7.1.11.5 Discussion

Each of these three technologies offers similar end results, but some offer a better development experience or a more feature rich pipline.

Poudriere is appealing because it is by far the most feature rich option available. It essentially offers a system for us to test and build our software all in one tool using the BSD Jails system. The consequence is that this package would need to be created manually each time we want to release an update to our software, or just to make changes to the package.

FPM is appealing mostly because this FreeBSD package will be a stretch goal. FPM will be able to take a Debian or CentOS package *as well as* produce a FreeBSD package. This will require minimal intervention, and as long as we can test that the package produced is correct we can use this for future iterations of the package. The downside is that this is not guaranteed to work correctly as it does not provide any testing infrastructure for the package produced, so some amount of manual testing will be required.

The last option is not appealing, but viable all the same. pkg-create can be used to create a package on FreeBSD. The tool is convenient to acquire, and allows us to create the package, but does not offer nearly as many benefits as the others.

7.1.11.6 Selection

For convenience we will use FPM. Once we have a Debian package creating a FreeBSD package should be smooth sailing. The solution is free, requires minimal human interaction, and allows us to create reproducible results (i.e., weather the package works or not is not determined by the package creator's coffee intake).

7.1.12 Windows Package

7.1.12.1 Option

There is three option technology for the windows install package. The first one is WiX Toolset, the second is EMCO MSI Package Builder, and the third is Advance Installer.

7.1.12.2 Goals for use in design

XZES40-Transformer may be packaged to run on multiple platforms. We want to create install package, so users do not need to install our package manually from source.

7.1.12.3 Criteria being evaluated

Although windows install package is our option work for our project, we want it move from other platform smoothly. We want it easy to learn how to use this technology, and we also want it free. The flow of install should be simple for user, so user can just install and run application quickly.

7.1.12.4 Comparison breakdown

- Cost: WiX Toolset is open source and free software however the advanced installer is a proprietary tool which must be paid for. EMCO MSI Package Builder provide free version for individual developer, but free version of package builder limit the functionality for create MSI package.
- Security: WiX Toolset is open source project since 2004. Even the Microsoft also uses WiX to create MSI package, so WiX tools should be the most safe tools than other tools. The security of Advanced installer is acceptable. The company is created since 2002, and other big company used this tools, such as Sony, Dell, etc. EMCO MSI Package Builder is same as the advance installer.
- Stable: WiX Toolset is open source project, and there is thousand of developer contribute on this project, so WiX Toolset is more stable than other open source alternatives.
- Learning Speed: WiX Toolset has a steep learning curve. It is better to understand the fact of Microsoft install package before using this tools. Advanced installer and EMCO MSI Package Builder provides GUI for user, it is easy to using and learning.

Technology	Description
WiX Toolset [**Wix_tool**]	• WiX Toolset is open source project • WiX Toolset is more stable and security than other tools • WiX Toolset have steed learning curve, but it is worth to learn • WiX Toolset is free for every one.
EMCO MSI Package Builder [**EMCO_MSI**]	• EMCO MSI Package Builder is not open source. • EMCO MSI Package Builder is stable, and security is acceptable. • EMCO MSI Package Builder is easy to learn, because it have GUI. • EMCO MSI Package Builder has free version for individual developer.
Advance Installer [**advanced_install**]	• Advance Installer is close source tools. • Advance Installer is stable and security. • Advance Installer is easy to learn. It has GUI, and forum support. • Advance Installer is expensive. No free version.

Table 16: Technology evaluations for system packages.

7.1.12.5 Discussion

The WiX toolset is a collection of tools which build Windows installation packages from XML source code. Traditional setup tools used a programmatic, script-based approach to be installed on the target machine but WiX uses a different method. It is uses an XML configuration file to describe the steps of the installation process. Microsoft also uses WiX with all its major software packages, like Microsoft Office. [**Wix_tool**]

EMCO MSI Package Builder is an installation tools designed for help developer create, maintain and distribute Windows Installer packages. [**EMCO_MSI**] It helps developer create MSI package automatically by using changes tracking technology, which is used to generate installation project data, or manually by using the visual editor.

Advanced Installer is GUI tool that can simply create Microsoft Install packages. Advanced Installer create and maintain installation package, such as EXE, MSI, ETC. It based on the Windows Installer technology. [**advanced_install**]

7.1.12.6 Selection

After comparing all of above methods we have chosen to use WiX as the solution to our Windows packaging problem. The first reason why WiX is my solution technology is that it is open source software. It a powerful set of tools available to create our Windows installation package.

The second reason is that it represents by source code rather than GUI. GUI may be easy for developer, but it also hided every thing behind the GUI. If we get some weired problem, we can't solve it via GUI software.

Third reason is that it can be completely integrated into our application build processes. When install progress start, other setup modification are made in parallel, so no vital information will be lost.

The fourth reason is that it use XML as the main language. We don't need spend time on learning how to use WIX.

The setup program will complete together with the application itself. This is not required by client, but if client want to different way to create windows install package, we can choose advanced installer as our alternate technology.

7.1.13 Conclusion

To summarize our findings:

Section	Author	Technology Choice
Research and Benchmarking	Zixun Lu	Xalan CLI, Altova
XML/XSLT Document Transformation	Shuai Peng	Xerces / Xalan C++ libraries
XML/XSLT Document Parsing	Elijah C. Voigt	Xalan C++
XML/XSLT Document Caching	Shuai Peng	Key-value C++ library
XML/XSLT Document Parallel computation	Zixun Lu	Linux POSIX threads
Web API	Elijah C. Voigt	Apache + Python CGI script
Web Interface	Shuai Peng	jQuery, HTML5, CSS3
Linux Package	Zixun Lu	FPM
Command Line Interface	Elijah C. Voigt	Bash proof of concept, Python final product
Windows Package	Shuai Peng	WIX
BSD Package	Elijah C. Voigt	FPM package creation toolkit

Table 17: Technology review findings.

7.2 Changes to technologies used

7.2.1 Interface (web) technologies

In the course of development we did not do a particularly good job of defining how we ought to create the web front-end. As a result it was thrown together fairly quickly a few weeks before the final deadline. The tools used in this component were not explicitly researched, but they were well thought out before development.

Specifically we used jQuery to perform asynchronous calls to the Apache CGI script, FileSaver.js and Blob.js (discussed further in the project's Markdown documentation) to download the resulting files to the users computer, and raw HTML5 and CSS3 to format the page. These technologies worked well and are industry standard tools for small front-end web pages like this one.

7.2.2 Cache

The cache ended up using a Key-value library provided to us by our client already in the Xerces/Xalan code-bases. This was ill-defined early on because we had not yet decided if we were going to write a cache system in-memory ourselves or use a different tool like Redis. The key-value scheme worked well and was very performed well in testing.

7.2.3 Parallel/Daemon

Although we discussed using higher levels of abstraction for creating forking processes. We ended up just using the system default Linux POSIX threads as this was the technology we were most comfortable with.

7.2.4 Vagrant

We expected to use VirtualBox for development but ended up using Vagrant as this was the tool one of the developers was most comfortable with and he was willing to put in extra effort to make this system robust enough for the project's needs. This ended up helping the team write setup scripts, used by Vagrant to setup a "one command" testing environment, which can be used at a later date for creating a system package.

The Vagrant environment automatically downloads the required operating system (Debian), installs the required packages, copies over the required configuration files (in Apache and systemd), and starts the web server. Users who want to contribute can run the service locally just need to install Vagrant and run ‘vagrant up’ to get a development environment; further change are automatically copied into the VM and they can compile code in the os by running ‘vagrant ssh’ and running ‘make’ in the VM.

This tool was used for its robust features, simple interface, and cross platform compatibility.

8 Development journal

8.1 Zixun Lu

8.1.1 Fall Term

8.1.1.1 2016-10-14

8.1.1.2 2016-10-21

8.1.1.3 2016-10-28

8.1.1.4 2016-11-04

8.1.1.5 2016-11-11

8.1.1.6 2016-11-18

8.1.1.7 2016-11-25

8.1.1.8 2016-12-02

8.1.2 Spring Term

8.1.2.1 2016-01-13

8.1.2.2 2016-01-20

8.1.2.3 2016-01-27

8.1.2.4 2016-02-03

8.1.2.5 2016-02-10

8.1.2.6 2016-02-17

8.1.2.7 2016-02-24

8.1.2.8 2016-03-03

8.1.2.9 2016-03-10

8.1.2.10 2016-03-17

8.1.3 Spring Term

8.1.3.1 2016-04-07

8.1.3.2 2016-04-14

8.1.3.3 2016-04-21

8.1.3.4 2016-04-28

8.1.3.5 2016-05-05

8.1.3.6 2016-05-12

8.1.3.7 2016-05-19

8.1.3.8 2016-05-26

8.2 Shuai Peng

8.2.1 Fall Term

8.2.1.1 2016-10-14

We meet our sponsor last week, and talk about the Problem Statement, and after that we all signed on that paper.

no problems so far.

Next week, sponsor will provide virtual machine which is Debian Linux system, so we can set up the development environment and ready for the program things.

8.2.1.2 2016-10-21

We get the virtual machine and some source file for our project from our sponsor(Steven Hathaway), and we also revise our problem statement.

No problem encountered so far.

We will contact sponsor to sign our revise problem statement, and finish our required document for our project.

8.2.1.3 2016-10-28

I write the requirement documents last weeks, and we talk to each other to make sure that we all understand our application. I also set up virtual machine and learn computer system problem.

Requirement documents is lots of pages documents, we spend time on what the specific requirement part.

We will work together to finish our requirement documents next week.

8.2.1.4 2016-11-04

We re-write the requirement documents. All of us sit together to think and type this requirement documents.

We have trouble with our performance requirement. I don't know how faster it should be, so I just guess what it is, and we decided to do benchmark.

I think that next week I will write about the technology review.

8.2.1.5 2016-11-11

Progress since last week

We discussed our project, and divide into 12 pieces. I am responsible for data transformation, web interface, cache, and Windows install package. Before Friday I have finished web interface part. I will finish other part during weekend.

I get trouble with website interface. I am not sure what is the definition of website interface. I will discuss with my teammate to understand what it is.

For next week, I want to think about the draft of our design document.

8.2.1.6 2016-11-18

Last week, we finished the technology review, and we have discussion that talk about design documents.

There is no more problem about our technology review.

We will have draft of design document before Thanksgiving.

8.2.1.7 2016-11-25

This week is short week, because of the Thanksgiving day. However, we finish the first draft of our design documents.

We have some problem about that some element in IEEE instruction documents. We will meet TA next week for clearly understand.

We will finish the design documents next week, and ready for the final presentation report.

8.2.1.8 2016-12-02

Since last week, our group meet together and write our design documents. We divide design documents as small section, and give each people individuals work. I wrote the cache, parser and website interface part.

There is lots of problem about the design documents. We are not really clearly understand some design viewpoint such as pattern. We asked the TA and understand what is the mean of this design viewpoint.

We plan to finish our final report for this term.

8.2.2 Spring Term

8.2.2.1 2016-01-13

During this winter break, I did research about the Redis cache manager program. And I also figure out how to run transformer in the virtual machine.

There is no big problem during the winter break.

I create the track tools in GitHub, so we are ready to start our development.

8.2.2.2 2016-01-20

I totally understood how to use xalanc++ library to convert xml file and xls file to new xml file. And I have complete part of our basic function for our program.

The xalanc++ API reference is hard to read. We need spend more time to search function that we need.

I will complete the whole basic function for our program. I mean we can transform a document successfully.

8.2.2.3 2016-01-27

I have finish the basic idea about our project.

Because I am not familiar with OOP development, so there exist lots of syntax error and type error. I need do lots of debug for fix this problem.

fix the syntax error and type error, and start work on cache.

8.2.2.4 2016-02-03

We have debug our main function, and make sure that our program can successfully transfer document.

There still bugs for xalanc++ using, we try to figure it out.

Get ready think about the cache part.

8.2.2.5 2016-02-10

During this week, we finally debug our program and complied it successfully. We finish our basically function of our program.

We just over the problem.

We will work on the cache part, and also the assignment of require document.

8.2.2.6 2016-02-17

This week our team come together and finish the assignment.

No problem so far

We need finish the cache function and continue our development.

8.2.2.7 2016-02-24

Last week I finish the cache code, and there is bug for delete. After I think, I decided that we don't need delete function for cache, we should use wipe function to handle the cache.

Delete get some problem, when we try to free data in cache, we got invalid pointer problem.

Elijah and me will merge cache and daemon together, and we will focus on the future work.

8.2.2.8 2016-03-03

This week I didn't do any code work, but I did more research on how to make the program run as parallel computation, because it's new to me, I need do more research on that. After I did research, I think that we should take POSIX Threads as our library to handle parallel computation.

So problem so far.

Maybe write some true code for our project.

8.2.2.9 2016-03-10

I wrote a prototype about the parallel computation, this is help us to understand how thread works.

The thread is create before run time, so I try to figure out how to make sure the thread is safely our requirement.

I will finish the parallel computation.

8.2.2.10 2016-03-17

We have meet the Steven to talk about the detail of the parallel computation, so I can make better plan and design for the parallel computation, and we come together to finish a draft of poster.

Parallel computation need a good plan to work.

Ready for the demo.

8.2.3 Spring Term

8.2.3.1 2016-04-07

We just come back and continue to work on our project. In this week, we merge the code what we have, and I was working on parallel computation.

I get trouble with the thread control, I need do research how many thread that we need, and how it work without conflict.

I want finish the parallel computation in next week.

8.2.3.2 2016-04-14

The second week, I have finish the parallel computation function for our application. To make sure that function runs correctly, we write a test to test it works or not. Finally, we get the thread working correctly without any crash or error. However it may not really safe, so we need more test in the future.

We meet Steven this week. Steven gives us a new requirement that we need pass the parameter to our transformer for transformation, so I have to do more research on the file parse. I need figure out how understand the element.

I want finish the parameter pass for our transformer, so we will finish every basic functionality that we need. The DTD files is also working, but so far it's not a vital problem for us.

8.2.3.3 2016-04-21

Last week I have finished the parameter passing back-end part. C++ file and our transformer is working.

So far, there is no problem for me, however, Elijah get problem on the front-end and web UI, so next week I will help him to finish the web UI.

Because May one is the last day that we submit our work, we want to come together to finish the documentation and do test and debug.

8.2.3.4 2016-04-28

Last week, out team did the last fix and improve with our application. We finally fixed the bugs that application has, and Elijah and I finish the documentation.

We get some website interface problems that web interface can not reply correct result from the back-end of the application.

During the weekend, we need do the last check for our program, so it should be ready to release. :)

8.2.3.5 2016-05-05

We have submit our project, so we basically didn't change anything last week.

No problem so far.

I think our team should do more test, so we make sure there is no problem during EXPO.

8.2.3.6 2016-05-12

Last week, our team come together and finished the progress report. We also discussed about the future work.

No problem so far.

We are ready for the EXPO!

8.2.3.7 2016-05-19

Since the last week, we have EXPO,and I just paper for that. I did exercise with my friends.

No problem so far, just need relax, and go to EXPO.

I am ready to write the final report.

8.2.3.8 2016-05-26

If I can redo my project, I would tell myself that you should paper everything early, so I can finish my project early and perfectly.

The biggest skills that I have learned is that solving a problem by Google or documentation. This project is really lack of documentation, lots of skill that I have learned come from the stack overflow and such that website.

Both of communication skills and solving problem skills I can use it in the future work.

Sure, I like my project, even it looks boring, but it really train our programming skills.

I learned lots of skills from our teammates, such that communication skills. I never talk to people such that client and high position people, but our teammate has experience, so I learned it.

If I am client of this project, I will be satisfied this project work, because everything that this team did is face my requirement.

If I can continue work it in the future, I would like do more test, fix bugs, and more error catch. Or we can have version 2.0 that it is support multiple platform.

I am little nervous during the EXPO day. This is my first time show my project in that environment. Everyone has interesting project and things, but our project is pure computer working. However, there still some people come to ask our question, this made me feel better. Thus I think that EXPO is a nice place to show our project. :)

8.3 Elijah C. Voigt

8.3.1 Fall Term

8.3.1.1 2016-10-14

During our first weeks in the course we met with our sponsor (Steven Hathaway) to discuss what he and the Apache Software Foundation wanted the project to look like. He had a clear vision for what the project would look like, how it would function, and what it would do differently than existing systems. The just of it being that a few layer of caching would be added to the transformation sever to speed up re-compilation of old documents (a common procedure).

Since last week we have finalized our Problem Statement document, confirmed it with our sponsor, and signed.

We have encountered no problems so far.

This coming week we will meet with our sponsor again to get setup with development environments. We decided on initially targeting Debian Linux as our platform, so we will be given VMs that are more or less the environment our application will be running on in production.

8.3.1.2 2016-10-21

This last week we continued editing our problem statement and met with our client to obtain a virtual machine for future development. The virtual machine was created by the client and included a slew of development tools for the package we will be developing and LaTeX development if we need.

There we no show-stopping problems encountered this week. One difficulty may be with the size of the virtual machine, which is almost 25GB, but once we have a stable development environment configured it shouldn't be an issue.

This coming week we will complete the Client Requirements document. We will also get a revised copy of our Client Requirements doc and Problem Statement doc signed and turned in on time.

8.3.1.3 2016-10-28

Since last week we have begun work on our Client Requirements document. We turned in a rough draft, but will have to make a lot of edits before turning in a final document next week.

There was a lot of confusion about our what our application is, how it will be implemented, what it will do, and its core purpose. To fix this our group had a meeting. We decided to have daily meetings to improve our document writing workflow. Future documents will be written with all three of us so we're on the same page. We will have daily hour+ long writing meetings.

We will spend next week re-factoring the Client Requirements document. Once we complete a draft we like we will get it signed and submit this document.

8.3.1.4 2016-11-04

This week we completed the client requirements document and had it approved by our client. He seemed impressed with the scope of the project.

Our client was too busy to have a meeting with us, but not too busy to sign the document, so it worked out.

This coming week we will begin work on the technology review. Now that we have a solid workflow for writing documents as a team I think this will be straight-forward.

8.3.1.5 2016-11-11

Since last week we have started working on our Technology Assessment document, and to an extent (at least internally) our design document. This document is not yet complete, but good work has definitely been put into it. Since bridging the knowledge gap I think that we've made good progress on unifying the design, and technologies being using in the project.

Our client has also supplied us with ample development scripts for our we have not started development these are not yet useful, but will be come winter term.

I don't recall any problems yet. I was unfortunately unable to participate in daily meetings because of travel on the holiday weekend, but I don't think this will cause tremendous problems.

This next week (starting Sunday) we will complete the technology assessment document and begin work on the design document. I hope to have a first draft of the design document by this coming Friday, but that may be wishful thinking.

8.3.1.6 2016-11-18

Since last week we finished our tech review and started the design document.

We were sort of down to the wire when we submitted the tech review, but got it in before the midnight on Monday deadline.

We started working on the design document but are a little confused about the exact format of the document.

This coming week, before thanksgiving, we will try to get a rough draft of the design document done.

8.3.1.7 2016-11-25

Since last week we have begun working on our Design document and we created a repository to store code by our Clients request.

We are confused about the exact format of the design document, we will ask the TA about this to get it cleared up.

This coming week we will finish the design document and make an initial draft of the term summary due during finals week.

8.3.1.8 2016-12-02

Since last week we completely re-wrote our Design Document and turned it in, unfortunately unsigned due to time constraints with out client. A large amount of time was spent restructuring the document to more closely fit the IEEE standard we were adhering to.

We also began work on the Progress report document in hopes of getting it done before the start of Finals Week.

The largest problem we encountered this week was having to re-structure our document. We misunderstood the IEEE structure for this assignment so we spent a lot of time re-writing the document after talking with the TA.

This coming week we will finish the Progress Report and Progress Report Presentation.

Over break we will also make headway on the actual assignment portion of the project, hopefully setting us ahead of schedule if all goes well.

8.3.2 Winter Term

8.3.2.1 2016-01-13

Over break I completed a large chunk of the project's structure. This included:

- Outlining the source code file-structure
- Added skeleton code for key classes, functions, and headers in the code.
- Added initial project documentation as well as code-documentation.
- Added a Makefile which currently compiles the project.
- Added a Vagrant virtual machine for lighter-weight development. This includes a setup script which can be used to provision a Continuous Integration system when we start using one.

Unfortunately I was the only member who was able to (or chose) to work on the project so there are a lot of decisions left to be made (changes to the design of the project including whether to use or a library provided by Steven Hathaway for caching, weather to use Boost for cross-os compilation)

This week I will complete the structure of the project so work can begin on transformation and caching.

The team-members who did not engage with the project over the beak will catch up by reading what has been written and documented, getting familiar with important libraries, and setting up their development environments.

8.3.2.2 2016-01-20

Since last week we went started going over what we need to do for the project in finer detail, learning our parts of the project, and learning the required tools we need to complete the project. We also started to plan in greater detail what and when we need to get tasks done? Tracking our progress better.

Shuai and Lu did not work on the project over the break so they are still catching up. Hopefully we won't be eventually behind and will eventually get back on schedule.

This coming week we will hopefully complete the basic transformation capabilities of our project, start bench marking competing products, and add continuous integration to pull requests.

8.3.2.3 2016-01-27

Since last week we have added Travis CI support and made progress in completing our basic functionality.

Shuai was unable to complete the basic transformation functionality I spent the majority of Sunday January 28 (I know this is due Friday but I was late and might as well include this info) making major refactors to contributions.

Lu has still not gotten any work done on this project.

This coming week I hope to get the basic transformation complete so that we can complete our alpha release on time.

Once we complete the basic transformation then there is a very short list of features to implement for the beta and final release. These are namely:

- Caching parsed documents.
- Daemonizing the application.
- Exposing the application to the web over an HTTP with a CGI script.
- Creating a Website for users to use the application.

Once those features are complete we will have finished the most important parts of our application. To finish any of those though we need to complete the core feature of document transformation.

8.3.2.4 2016-02-03

We have made very little progress since last week. As mentioned previously, I worked for most of a sundae debugging the code that Shuai worked on. Since then we have not figured out a solution to the bug.

Lu has not been communicating nearly enough with the group, and we have not solved this bug. Once we fix a show-stopping bug we should be able to start making progress individually on the application.

I have asked Lu to work on the bug. If he is not able to fix it I will do my best with Shuai to fix the bug before the end of next week so we have a working alpha release.

8.3.2.5 2016-02-10

This week our group (Shuai and myself) finally tackled and completed the core of our project.

The biggest problem was completed, but we still have a long way to go. Our biggest problem now is finishing all of the work we have ahead in the time we have available.

Strictly speaking though we didn't encounter any new problems, just fixed existing problems.

That said Zixun Lu has still not completed the benchmarking for the project. This is a problem as we will not be able to assess the success of our project without that information.

This coming week I will try to start and finish the daemon-ization of the program, but with the progress report assignment most actual development is practically suspended.

8.3.2.6 2016-02-17

Since last week we have made minimal progress on the application. I have begun work on demonizing our application.

Process daemonizaion was not a problem we thought out well enough so a lot of planning has to happen pretty late in the game for that part of our application to work correctly. I think I know a way to execute the idea using best practices, but so far it's not been easy.

Tomorrow (Sunday) we will meet and develop for a full day to hopefully power through some problems we've been having. I am not optimistic that we will complete anything meaningful. I hope to be proven wrong.

8.3.2.7 2016-02-24

Since last week I made progress on and eventually merged the daemon code. I also helped Shuai Peng with the Caching code.

The daemon works fairly well. It has not been battle tested, nor has it been designed to handle errors or problems well. These have not cause any headaches yet.

This sundae Shuai and I will complete and merge the Caching code. I will begin work on the Web API. We will also demo what we have this Thursday.

8.3.2.8 2016-03-03

Since last week we have not made substantial technical progress. After merging the Caching code Shuai and I have focused on other coursework. I personally have been researching how to write the CGI script for our application, but not as much time has been spent as I would like.

We also met with our was productive and expectations were tempered as we have been missing deadlines. I still believe we will be able to get a working prototype completed by our deadline, however it will not be nearly as polished as I hoped and expected given the complexity of the project.

Zixun Lu has not been able to complete his portion of the project and we are as of yet far past our deadlines for the Benchmarking spring. He has been reassigned to other parts of the project which do not require C++ coding (benchmarking, website frontend ben b, and the system package) as this seems to be intimidating for him.

This coming week I will finish a prototype, and hopefully merge, the CGI script portion of our project as well as any setup scripts to start an Apache scripts server.

This will include:

- The actual Python CGI script(s).
- Setup scripts for installing and enabling an Apache server CGI service.
- Possibly some systemd services.

Actually, may I go on a quick tangent about how useful a systemd service can be for this project? Using systemd we can limit the CPU and RAM usage of an application, thus giving us a 'first line of defense' against runaway cache. Not that I think that is a likelyhood, however it is a nice 'stopgap' just in case. We can set the application, in the init system layer, to restart and wipe the cache whenever it reaches some threshold. A more ideal solution would be to garbage collect the cache every so often so we don't need emergency safeguards, but if it works it works!

8.3.2.9 2016-03-10

Is it already week 9? That term flew by...

Since last week we haven't made a terribly large amount of progress. I've gotten the CGI script working so we can (hopefully soon) start communicating with the outside world and running jobs over the 'net.

Originally I tried implementing the CGI script using 'fastcgi' which was touted as being the way to do this, but it didn't work. Our client helped out and told us to just use the regular old boring 'cgi'.

I am very close to getting this working, so I'll get that done before we need to demo.

8.3.2.10 2016-03-17

Good news! We finished the demo!

As the clock struck 10:30 I finally got the demo working. You could upload two files and get the transformed output in your browser. It took a long time, but it was totally worth it.

8.3.3 Spring Term

8.3.3.1 2016-04-07

We encountered many problems on the way, and have many more problems to come, but the big ones related to this were:

1) CGI scripts are non-trivial to setup.
2) Error handling and debugging cgi script was non-trivial.
3) Slightly tweaking (or in some cases entirely re-writing) parts of the code was time consuming.

This coming week I am going to finish finals and take a brief vacation, but starting next term I will spend a substantial amount of time fixing up our pull codebase for Expo and making sure it is ready for the client to take over. This includes documenting the codebase, 'sanding the edges', and any tools Steven needs to own he project when we leave the project.

8.3.3.2 2016-04-14

Since last week we have completed benchmarking of our application (or at least a first draft) and implemented a prototype of parameter passing (https://github.com/XZES40/XZES40-Transformer/pull/38).

The biggest problem we encountered was that we had to hack together an environment to actually do the benchmarking. A complete application would allow us to upload header files for our benchmarked XML documents and pass parameters, but we have yet to complete these features.

This coming week I would like to complete the parameter passing and integrate it into our website I will add the web interface and form processing / sending this week.

Next week we will try to add dependency file process. This will be similar to the parameter passing feature, but adding files to the build job instead of key=value parameters.

8.3.3.3 2016-04-21

Since last week I have begun working on passing parameters to the application through the web UI.

The hardest part of this is that I have to format the data in such a way that the input form data is sent as JSON to the end CGI script. This can be done with JQuery, but I'm not a frontend person so it's taking longer than the rest of my components did.

This coming week I want to merge parameters and a big docs push. We should be code-complete by friday the 28th (my birthday!) so I'll make sure we have something presentable by then.

8.3.3.4 2016-04-28

Since last week we have made leaps and bounds.

- We (Shuai and I) fixed a lot of last minute bugs and added some much needed revised documentation.
- I made our website dynamically submit transformation jobs and load the response content.
- I learned JQuery (for the website).

We ran into a few bugs, especially around form processing and displaying results to the user on the frontend. Thankfully we fixed most of those bugs and by Sunday we should have all of that ironed out for Code Completion.

This coming week we are going to merge the last pull requests we have for Code Completion and begin working on our demo and written documents for the course.

As I mentioned last week, today is my birthday, so I'm going to take a day off. We got a lot done and really brought it all together at the last minute. Now we just need to put a bow on it.

8.3.3.5 2016-05-05

Since last week we completed our code and poster and submitted both of those. We did not work on the code for our project, favoring instead to work on other homework to get ahead for the end of the term.

We had a few hiccups with our code, trying to merge a lot of changes together at the last minute. Thankfully it passed the smoke tests so we were feature complete.

This coming week I would like to add more tests to the application, however as a team we will probably not do this.

8.3.3.6 2016-05-12

Since last week we finished a progress report (ahead of schedule!) and started prepping for Expo.

No problems have been encountered this week.

Up next: Expo!

8.3.3.7 2016-05-19

Since last week we just made sure everything was ready for expo.

We didn't encounter any problems at expo. Thank god.

This coming week I will get the three short writings out of the way if possible.

8.3.3.8 2016-05-26

My biggest regret in the course of this project was over-engineering solutions which did not need to be made. The biggest example of this was when I spent all of winter break outlining the code we needed for our project, creating skeleton code for the majority of our C++ work, and in the end a large swath of that was removed to get the project done.

This was a blessing and a term of course learned a good lesson, and had a good understanding of our project going into winter term, but it was a blow to the ego when a bunch of my Hit was wasted.

The biggest skill I've learned over the course of this project was time management, and allocating work failrly. I tend to feel an urge to over-work myself when others are underperforming. This is unfair to myself and the people under-performing. So my new skill is probably something like "tend to your own garden".

The biggest skills I can use in the future on the technical side are some nifty javascript skills and some nice Apache system admin skills.

On the non-technical side I've gotten very good at managing a small team and I've learned a lot of lessons the hard way. As mentioned above, I have a new understanding of how to allocate jobs and stick to that allocation. Don't over work yourself just to get the job done, hold those who agreed to a job accountable. Otherwise you'll pull your hair out trying to get somebody to do a job you're already doing for them.

I am glad to have contributed to the Apache Software Foundation I am also glad that this project is over.

I learned from my teammates many lessons about management, and how as a manager you should meter your expectations a lot. Don't expect anything from your teammates, make everything explicit, and make sure you stick to your job.

I believe the client is satisfied. Although we did not know this going in, this was more or less a prototype project and to that end we definitely finished the prototype.

I will volunteer myself as a contributor for this project going forward. I want to support the Apache Software Foundation and this is a pretty simple way to do that.

At Expo we were surrounded by really exciting project so we didn't get much attention at Expo. That said I did meet someone from Nvidia who was very excited about our project and told me to send her an email when I was looking for a job, so that was a success!

9 PROJECT DOCUMENTATION

The following section includes the project's documentation.

The document has not been reformatted to best preserve the original content.

If you would like to view the documentation in it's 'pretty print' formatting, visit https://github.com/XZES40/XZES40-Transformer/

The headings of each section here reflect the original file's location in the directory structure of the source code. The documentation co-mingles with the source code.

Each source file also includes documentation blocks, which are not included here.

9.1 README.md

```
# XZES40-Transformer

*This repository contains the Code, Code Documentation, Tests, Examples, and Build/Dev infrastructure
    for the Oregon State University + Apache Software Foundation Computer Science 2017 Capstone project
    .*
For more information about this project, see the project's [LaTeX document repository][capstone-repo]

## Build status

- Linux: [![Build Status](https://travis-ci.org/XZES40/XZES40-Transformer.svg?branch=master)](https://
    travis-ci.org/XZES40/XZES40-Transformer)

## About the project

This project's purpose is to create an XML/XSLT transformation server which is

- *Fast*
- *Cross-platform*
- *Packaged for your OS*
- *Open Source*

It achieves these by:

| Feature                                                 | Status              |
| ------------------------------------------------------- | ------------------- |
| Processing documents in parallel.                       | **IMPLEMENTED**     |
| Using an in-memory cache to store processed documents.  | **IMPLEMENTED**     |
| Using [FPM][fpm] to package the application.            | **NOT IMPLEMENTED** |
| Using the [Apache Webserver][apache] to handle requests.| **IMPLEMENTED**     |
| And of course developing in the Open!                   | **IMPLEMENTED**     |
```

For a complete guide on the *design* and *purpose* of this project refer to our [LaTeX Documents Repository][capstone-repo].

A what transformation server?

XML is a standard format for storing data, like JSON or YAML.
XSLT, or XML Style Sheets, are used for performing transformations on data in XML sheets.
For example:

Problem

- You have an XML spreadsheet with information about all cars sold at a used car lot.
- You want to know how many *red* cars were sold between 2000 and 2015.

Solution

Create an XML stylesheet (XSLT document) which specifies "if the <color> is red, and the <date sold> is between 2000 and 2015, include it in the output" (in psuedo XSLT).

The XZES40-Transformer application is used to perform these transformations over the internet so one can process data without installing heavy XML transformation applications locally.

How do I use this?

The easiest way to use the application is to run a small Virtual Machine (VM).
The application is designed to be run on Apache, but it is not encouraged that you install, run, or develop the application on your personal machine.

The tool the development team uses to run the application in a VM is Vagrant.
Vagrant is a Command Line Interafce (CLI) interface for pulling, provisioning, and running VM with arbitrary back-ends.
This means you can use the same commands to run a VirtualBox, VMWare, or Docker machine.
Vagrant runs on Linux, BSD, Windows, and MacOS, so it is highly accessible for development.

Once you have Vagrant installed and setup on your system you should be able to run `vagrant up`.
Once the machine is setup it will give you a URL to visit in your local web browser.
Visit that page and you'll be using the application!

For information on running the application with Vagrant, see the [`vagrant` directory's README file][vagrant-code].

Where is everything?

There are three components to the XZES40-Transformer project.

Component	Description
`transformer`	Core of the project. Actually performs XML transformations.
`cgi-glue`	Bridge between the transformer and web UI.
`frontend`	Web interface for the application. Accepts XML transformation requests, displays

```
    results to user. |

Each of these components can be found under the ‘xzes‘ directory.
Each component has an explanatory README which explains what it in greater detail is and how it works.


## How can I test?

In the directory ‘xzes/transformer/test/simple_test_file‘ we have provided a simple test script that you
     can run on your local system which connects to the server running via Vagrant..
Just run ‘./simple_test.sh‘ and then you should see HTTP 200 OK responses and transformed XML documents
    ..

## Contributing

If you have interest in contributing to this project, **great**!

Please create an [issue][issues-url] and see if the developers respond.
If you have a fix for your issue, make a pull request by forking the project, doing your work on your
    fork, and clicking the ‘Create a Pull Request‘ button once you push your changes to that fork.

We have a [Vagrant Box][vagrant-code] which is probably your best way to test code-changes.
See the ‘vagrant‘ directory for details about that work-flow.

## A note about names

There might be a bit of confusion about what is named what and why:

- The project is called XZES40-Transformer because the Capstone group name is XZES40 (a combination of
    our names, our group number, and the letter X).
- The application is just called ‘xzes‘ because programs shouldn’t have numbers in their name.
- The transformer daemon is called ‘xzesd‘, and it’s systemd service is called ‘xzesd.service.‘
- The CGI script used by apache is called ‘xzes.py‘.
- The component that runs between Apaache and the daemon is ‘cgi-glue‘ because it’s just enough code to
    get ‘xzesd‘ exposed to the internet.
- The frontend is called ‘frontend‘ because that is an intuitive name.

*Most* things, unless they are the group or project, are just ‘xzes‘.
The project as a whole is ‘XZES40-transformer‘.
There are no components called simply ‘XZES40‘, although during development some components *were*.

## License

This software is licensed under ‘Apache 2.0‘. Read the ‘LICENSE‘ file for more information.

[capstone-repo]: /XZES40/cs-capstone-project
[issues-url]: /XZES40/XZES40-Transformer/issues
[fpm]: /jordansissel/fpm
[apache]: https://httpd.apache.org/
```

```
[vagrant-code]: vagrant/
```

Listing 3: XZES40-Transformer project REDME

9.2 scripts/README.md

```
# Scripts

This directory includes scripts used for development that didn't really fit with a specific part of a
    project **or** had multiple purposes.

The scripts are exclusively written in Bash and focus on operatics.

The `ci` directory includes scripts intended for Continuous Integration.
If you are interested in how a component is used, see if it has a script associated with that.

The CI system we intended to use heavily was TravisCI.
Most of the `ci` scripts do not get used except for the one that builds the software (`transformer.sh`).
That said most of the scripts are [mostly] accurate, just not robust enough for an automation build
    system.
```

Listing 4: Context for the scripts directory.

9.3 xzes/README.md

```
# XZES Projects

These components include:

- `transformer`
- `cgi-glue`
- `frontend`

Each project includes it's own README which self-describes the component, what it does, and how it works
    .

## xzes.conf

This file is the Apache config for the XZES40-Transformer application.
Feel free to edit this to change the port the application runs on, add security certificates, pretty
    much anything you would do in an Apache HTTPd config.

## xzesd.service

This is the systemd service file for xzesd.
It automatically restarts the application, manages forking the process into the background, and
    aggregates xzesd stdout to the Linux logging service.
```

Listing 5: xzes code README

9.4 xzes/cgi-glue/README.md

```
# cgi-glue

This component of the project is a python CGI script which accepts HTTP requests via Apache, processes
    them into a format that the xzesd daemon can read, saves uploaded files, waits for the
    transformation to complete, and responds with the output.mdt from the daemon.

‘xzes.py‘ is the meat of it. Read the comments or more information.

‘simple_test.sh‘ is used to test to see if the application is running correctly. Pass it an xml and xsl
    file (in that order) and see if it responds correctly.

‘benchmarks.sh‘ was used to benchmark the speed of the application for grading purposes.

This is essentially treated by the frontend as a *very* simple.mdt-based web API.
A form with two files (‘xml‘, ‘xsl‘) and parameters (‘{key: val, key2,val2}‘) are accepted, processed,
    serialized, and sent to the daemon over a local network connection (‘localhost:40404‘).
```

Listing 6: Overview of the Python Apache CGI component.

9.5 xzes/transformer/README.md

```
# Transformer

This is the C++ application which transforms and XML and XML style-sheet documents into an output XML
    document.
This transformation is requested over a local network connection on ‘localhost:40404‘ via an Apache CGI
    script found in ‘cgi-script‘ (located in the parent directory).

## Usage

### Building

Once you’re in a VM with the correct packages installed (check the ‘vagrant‘ directory for a reference
    as to what those packages are), you can build the project.

‘‘‘
(transformer/)$ make
[... lots of verbose output ...]
‘‘‘

### CLI Usage

Now that you’ve built the project you can *use* it.
How fancy.

‘‘‘
user@host:transformer/$ ./build/xzesd &    # This runs the daemon in a background process
[2] 14876
user@host:transformer/$ ./build/main.py ‘pwd‘/examples/simple.xml ‘pwd‘/examples/simple.xsl /tmp/example
    .xml
user@host:transformer/$ cat /tmp/example.xml
<?xml version="1.0" encoding="UTF-8"?><out>Hello</out>
```

```
## Development

To develop the application you will need

- A Debian Linux host (probably a virtual machine).
- The C++ dependencies listed in the ‘vagrant‘ directory in the ‘setup.sh‘ script.
- The ability to host a website locally.

### Using Vagrant

Vagrant us a convenient tool for using a portable Virtual Machine.
Our development environment depends heavily on Vagrant.
It automatically runs a setup script which installs the dependencies, builds the binary, installs apache
    , sets up the daemon service, and starts the webserver to be accessed on your local virtual machine.

```
(XZES40-Transformer/)$ cd vagrant
(XZES40-Transformer/vagrant)$ vagrant up
Bringing machine ’default’ up with ’virtualbox’ provider...
==> default: Checking if box ’debian/jessie64’ is up to date..
[...]
==> default: Vanilla Debian box. See https://atlas.hashicorp.com/debian/ for help and bug reports
(transformer/vagrant)$ vagrant ssh
[...[
vagrant@jessie:~$ cd xzes/xzes/transformer
```

See the file ‘vagant/README.md‘ for more information about the Vagrant development environment.

Even if you are not using Vagrant, the README and setup script should help you start contributing if you
     want to do that.

Structure

This project is organized according to a [’better than nothing’ standardized structure][cpp-project].
This looks as follows:

```
project/
|-- build
|   ‘-- generated binaries (*.o, main)
|-- doc
|   ‘-- documentation (*.txt, *.md, *.rst)
|-- examples
|   ‘-- code examples not necessary for building but helpful
|-- include
|   ‘-- headers (*.h, *.hpp)
|-- lib
```
```

```
|   `-- external dependencies (*.h, *.hpp, *.c, *.cpp)
|-- Makefile
|-- src
|   `-- code (*.c, *.cpp)
`-- test
    `-- unit tests (*.c, *.cpp)
```

In addition to this structure, each source code file has

- A `.hpp` file in the `include` directory which outlines the classes, methods, and functions for that component.
- A `.cpp` file in the `src` directory which implements the classes, methods, and functions in that component.
- A `.md` file in the `doc` directory whichcovers the design, co.mdt, and oddities of that component.

All components in the project are compiled into `.o` files in the `build/` directory.
The entrypoint binary is compiled to `build/main`.

Documentation

Documentation about the code is written in the `doc` directory but the source files are also self-documented.
For reference the `doc` directory tends to talk about the high-level concepts (what a compent is supposed to achieve and how it fits in) while the code documentation is targeted at helping developers find bugs and implement features.

[cpp-project]: http://stackoverflow.com/questions/10782554/how-to-organize-a-c-project#10782577

Listing 7: Documentation about the transformer as a whole.

9.6 xzes/transformer/test/simple_test_file/README.md.md

```
# How to use this test?

## In the command-line

To auto-run the tests run the following commandin this directory:

```
./simple_test.sh
```

This can be run in the virtual machine, or on your local system.
It depends on the application being avaliable on `192.168.33.22:8000`.

In the browser

Once the application is up and running using the Vagrant setup script, go to the address `192.168.33.22:8000`.
For each pair of files in this directory (`sample1.xml`+`sample1.xsl`, etc), upload the pair, click 'Transform Files' and when the button becomes available click 'View Raw XML'.
```

```
Fore each sample you can also add parameters.
The accepted parameter keys are `param1`, `param2`, etc. up to `param5`.
The key must be one of those but the value may be anything you want.

`sample1` does not use parameters, but all of them can accept parameters.

Click the 'View Raw XML' button to refresh the document preview.

Click 'Download Output' to download a copy of the transformed file.

## How do I know it works?

In the CLI you should see 200 OK response output form `curl`, and transformed XML documents.

For the web-browser testing you should see the output of the transformation in the preview.

If you upload a *non* XML/Style Sheet you should see an error message.
```

Listing 8: Usage information for the simple test / smoke test.

9.7 xzes/transformer/doc/lib.md

```
# Lib

The library (`lib`) is used for miscelaneous functions used across the project.
This is kept as small as possible since any large objects and groups of similar methods will be broken
    out into their own components (like the Document class).
```

Listing 9: Documentation about the general purpose library code.

9.8 xzes/transformer/doc/README.md

```
# Documentation

These documents are designed for developers to get an understanding of what the application is, what
    each file/component is doing, and in some cases why it was designed that way.

Read at your own peril.
```

Listing 10: Context for the documentation directory.

9.9 xzes/transformer/doc/daemon.md

```
# Application Daemon

**NOTE** In production the daemon is called `xzesd` and is stored on disk at `/usr/local/bin/xzesd`.

In order to preserve state in the in-memory cache, XZES has a long-running daemon.
This could have been implemented a number of ways: Redis, a database, carrier pigeons.
We chose an in memory cache as the volatility is not hyper detrimental to the goals of the application (
    convenience, then speed).

Here is roughly how that daemon works and how it recieves jobs.
```

```
 +----------------------+
 |                      |
 |   Main Executable    |
 |                      |
 +--^----------------+--+
    | network socket |
+-------------------------------------------------------+
|   |                |                                  |
|   |                |                                  |
|   |                |         +--------------------+   |
|   |              +------------>                    |   |
|   |                           |   Transformer      |   |
|   +---------------------------+                    |   |
|                               +----+--------^------+   |
|   Daemon Process                   |        |          |
|                                    |        |          |
|  +---------------------------------v--------+------+   |
|  |                                                |   |
|  |   Cache                                        |   |
|  |                                                |   |
|  +------------------------------------------------+   |
|                                                       |
+-------------------------------------------------------+
```

Usage

The daemon is included as a systemd service in the application's codebase (`xzesd.service`).
This service is setup and enabled in the `vagrant` setup script.

The daemon can be started manually with the following commands:

```
$ make              # Build the daemon
[... successful build ...]
$ ./build/xzesd &  # spawns a daemon process in the background
[1] 14877
```

... and tested like so:

```
$ ./build/main.py `pwd`/examples/simple.xml `pwd`/examples/simple.xsl # check that the dameon is running
<?xml version="1.0" encoding="UTF-8"?><out>Hello</out>
```

This assumes that the application is not running with the web-app component. To test that see the `cgi-bin` directory's README.

```
## The Daemon process startup

1. When the `daemon` binary is executed it sets up a long running process and goes into a state of
    cybernation.
2. This process starts by:
  - Opening and listening on a `localhost:8000`
  - Allocating a block of memory for the cache.
  - Spawning a set of threads on which transformations may occur.

## Spawning a Transformation

1. When the `main.py` script recieves a request it is passed two files, input xml and xslt file-paths,
    as well as some parameters.
   This is either done via the web API (tested with `simple-test.sh` or a CLI directly talking to the
       daemon... probably the first of those two.
2. The `main.py` script opens a local socket connection with the daemon process.
3. The request is passed in a tuple with the following format: `"request-id#,input-filename.xml,input-
    filename.xsl,output-filename.xml,{param1:value2},{param2:value2}..."`.
4. The daemon process the arguments checking for errors.
5. The daemon process spawns a thread and calls the `transform_documents` method in that thread.
6. Once the thread completes the transofrmation it returns a status code which is propagated up to the `
    main.py` script and returns to the calling process.
```

Listing 11: Documentation about the application daemon code.

9.10 xzes/transformer/doc/main.md

```
# Main

**NOTE** This file has been deprecated in favor of `daemon.cpp`.
Please read the daemon document instead of this one.

Main is the entrypoint for the entire application.
It essentially just parses command-line arguments and either passes those to the Transformer or yells at
     the user for using the application wrong.
```

Listing 12: Documentation about the 'main.cpp' file.

9.11 xzes/transformer/doc/cache.md

```
# Cache

The caching component of XZES40-Transformer is designed to create an easily usable Caching API for the
    Transformer and Document components.

Cache uses the XercessC keyList to manage the stored state.
The data structure of the keyList is link list.
Although link list is slower than hash map, it was used becuas it was provided by the XercessC Library.

## API
```

```
The current interface looks like this:

'''cpp
#include <lib.hpp>
#include <keylist.hpp>
#include <cstdlib>

        class Cache {
        private:
                KeyListEntry *theList;

        public:
                Cache();
            bool search(id_t);
            doc_t* get( id_t );
            int set( id_t,doc_t*,uri_t );
            int print_name();
            int print_id();

    };
'''

To clarify:

- `Cache()` which is the default constructor.
- `bool search(id_t)` will search the content inside of cache, return true if there is file exist.
- `doc_t* (id_t)` return the content from the cache.
- `int set(id_t,doc_t*,uri_t)` Set the content to cache. id_t is id, uri_t is name, doc_t is content.
- `int print_name()` is a debug tools for printing all content from the cache.
- `int print_id()` is a debug tools for printing all id from the cache.

# Usage

The only component of the Cache which is used is for storing Documents.
When a document is created it is parsed and cached automatically, given an input file.

The data flow is as follows:

1. The `Document` constructor sets the URI and hash for the given document.
2. The `Document` constructor checks if the document is already in the cache for the hash key.
    - if true, run cache.get and document.set_content for `Document` class.
    - if false, run document.complie(), and cache_set() for `Cache` storing.
5. Done
```

Listing 13: Documentation about the caching code.

9.12 xzes/transformer/doc/main.py.md

```
# main.py

Main.py is a test python script for ensuring the application daemon communicates correctly over the
    local netowork.
```

```
This assumes that the daemon us running on `localhost:40404`.
```

Listing 14: Documentation specifically about the proof of concept 'main.py' script.

9.13 xzes/transformer/doc/keylist.md

```
# KeyList

The keylist file contains functionality used by the cache to store state, specifically parsed documents,
     in-memory.
```

Listing 15: Documentation about the keylist library used for in the caching.

9.14 xzes/transformer/doc/transform.md

```
# Transform

The `transform_documents` method wraps the use of the Transformer class.

## Transformer class

The Transformer's job is to perform the actual application's purpose: transforming an XML document with
    an XSLT document and writing that to disk or printing it to the screen.

### Members

The three main members of the Transformer are:

- `Document xml`
- `Document xsl`
- `Document out`

These are created with the Document class initializer outlined in the Document ... err... doc.

### Methods

When the Documents are created they contain three components themselves `id`, `uri`, and `contents`.
The `contents` is the part we care about.

Using the Xerces and Xalan libraries we perform a DOM transformation with `xml.contents` as the XML
    component of the transformation, `xsl.contents` as the XSLT component and `out.contents` as the
    output.

### Caching output documents

Once the `xml` and `xsl` documents are initialized the cache gets requested for a document with the id `
    xml.contents + xsl.contents`.
This is a unique identifier at which all output documents are stored.
If the output document already exists then we bypass the entire transformation process and return this
    cached output document.
```

```
Once saved the output document is saved to the cache by it's own `id` **and** by the `id` `xml.contents
    + xsl.contents`.

Only output documents have a double-length id so there should be no collisions.
```

Listing 16: Documentation about the transformation code.

9.15 xzes/transformer/doc/parse.md

```
# Parse

The file `parse.cpp` contains functionality for parsing input recieved from the user via the python CGI
    script into a struct which is easy to operate on.

That input is in the following format:

`"job_id,xml_file_path,xsl_file_path,output_file_path,key1,val1,key2,val2,key3,..."`

This means that the daemon recieves what essentially amounts to a stream which is then parsed.

The `key1,val1,key2,val2` are tuples of key=val pairs passed to the transformer for custom
    tranfsormations.
```

Listing 17: Documentation about the parsing code.

9.16 xzes/transformer/doc/parallel.md

```
# Parallel Computation

Parallel computation is how XZES40-Transformer hold the faster transformer performance.

Parallel computation will hold multiply threats for XZES40-Transformer application.
We will using Unix-socket lock function to handle the time conflict when we have mutiply file at the
    same time.

Here is how Parallel computation run.

1. First we have CGI script receive request from the front side.
2. CGI script may receive mutiply request at the same time, but they only can write at one time.
3. Daemon receive request and have mutiply thread to transfer the document, but only can return
   a transfered file at one time for one request.

---


## Here is a map how parallel looks like:


                        +------------------+
                        |  daemon running  |
    +--------+          |    multiply      |
    |        |  request1 |     thread       |
```

```
    |  CGI   |  request2 |                  |
    | script +----------->                  |
    |        |  request3 |                  |
    |        |           |                  |
    |        |           |                  |
    +---^----+           |                  |
        |                +---------+---------+
        |                          |
        |                          |
        |                          |
        |                          |
        |                          |
        +--------------------------+
                  return
```

Here is the time slot represent how parallel looks like:

Daemon:

Request 1 received ->> create thread ->> do transformer ->> lock socket ->> return file ->> unlock socket

Request 2 received ->> create thread ->> do transformer ->> __wait,locked socket__ ->> lock socket ->> return file ->> unlock socket

CGI script:

Request 1 send ->> lock socket ->> write to the daemon ->> unlock socket ->>receive return file

Request 2 send ->> __wait,locked socket__ ->> lock socket after detect unlock ->> write to daemon ->> unlock socket ->> receive return file

Listing 18: Documentation about the parallel transformation code.

9.17 xzes/transformer/doc/document.md

```
# Document

Document is a C++ class with three main components, as well as methods which operate on those components.

Documents are stored and retrieved from the Cache and are operated on by the Transformer.
While we could use the Xerces and Xalan object natively, this wrapper allows for a seperation of concerns.
The Cache can worry about storing an arbitrary `class`, the Transformer worries about transforming the `Document.contents`, and the each component can be developed (mostly) seperate.

## Members
```

```
Here is the rough outline of the variables in a Document:

- `uri`: The file path or remote address that the contents of a Document are stored.
- `id`: Is a **unique** ID, probably created by hashing some safe metadata about the source file.
- `contents`: Are the **parsed** XML or XSLT Document.

The raw contents of a file are not stored, only the parsed version.
This may change going forward.

## Methods

The Document class has a few methods:

- `Document()` is the initializer.
- `set/get_uri()` set and get the URI respectively.
- `set/get_id()` set and get the ID respectively.
- `set/get_contents()` set and get the contents respectively.
- `to_str()` processes the `contents` member and outputs and XML/XSLT document (useful for `output`
    documents).
- `to_file()` processes the `contents` member and outputs the XML/XSLT document to a file (useful for `
    output` documents).

## Dataflow

The Document class is the component of the project which interfaces with the Cache the most.
Here is why:

1. For a Document to be created, it must be passed a URI (file path or HTTP address).
2. The URI is parsed and basd on that an ID is created by either hashing the file contents or hashing
    metadata about the remote URI.
3. The construcor then checks to see if the Document ID is a key in the Cache.
    - If the Document is *not* in the Cache it is parsed and stored in the Cache.
    - If the Document *is* in the Cache the value of `contents` is filled with the Cached value.
4. Once the Document is initialized it is returned (probably to the Transformer).

The Document component was broken out into it's own file as it pertained to both Caching and
    Transformation and seemed complicated enough not to need it's own source file.
```

Listing 19: Documentation about the document class abstraction.

9.18 xzes/frontend/README.md

```
# xzes frontend

This is the frontend web interface for xzes40.
It is served via an Apache httpd server.
The Apache conf can be found in this component's parent directory.

## Dependencies
```

```
The frontend depends on [FileSaver.js][filesaver.js](git commit 4db4a78) and [Blob.js][blob.js](git
    commit 079824b) , both licensed under the MIT license, for downloading transformed files.

To upgrade these dependencies:

1. Visit their GitHub pages
2. download `FileSaver.js` and `Blob.js` respectively.
3. replace the current copies of those files in this directory with the updated versions.

The frontend also depends on JQuery, which is resovled by a Content Delivery Network (CDN).
If you need to use a local copy of JQuery, please consult the JQuery documentation for this task.
It roughly looks like:

1. Downlaod JQuery 3 (latest, full, minified).
2. Place it in this directory.
3. Replace the JQuery `script` tag with a refernce to the downloaded copy.

[blob.js]: https://github.com/eligrey/Blob.js
[filesaver.js]: https://github.com/eligrey/FileSaver.js/
```

Listing 20: xzes frontend code REDME

9.19 vagrant/README.md

```
# Vagrant development environment

This development environment requires the installation of [Vagrant][vagrant]
(and probably [VirtualBox][vbox]).

## About the dev environment

The Vagrant box includes the following:

- Creates a Debian Jessie virtual machine.
- Mounts the repo to the `/xzes` directory (which is symlinked to `/home/vagrant/xzes`).
- Installs the `xerces` and `xalan` libraries.
- Sets up the Apache `xzes.py` script and VirtualHost.
- Enables and starts the `xzesd` systemd service.

This is designed to a minimal virtualbox only consisting of what is
*necessary* for the XZES application to run.
Nothing more, nothing less.

## Usage

1. Install Vagrant and a virtual machine runner (VirtualBox for instance).
2. In a shell navigate to this directory.
3. Run `vagrant up` to start the virtual machine.
   If everything works correctly this command will output debug information and exit successfully.
   **READ THE OUTPUT OF THIS COMMAND**
4. Run `vagrant ssh` to login to the virtual machine.
5. On your host OS, run `vagrant suspend` to put the machine in hibernation.
```

6. On your host OS, run ‘vagrant destroy‘ to destroy the machine.
   This will not delete any data on your host machine, only data stored exclusively on the machine.

NOTE: You may need to install the vagrant plugin ‘vagrant-vbguest‘ if you get an error about mounting directories:

```
$ vagrant plugin install vagrant-vbguest
```

Once the vagrant machine is up, i.e., just running ‘vagrant up‘ successfully, you should be able to go to ‘http://192.168.33.22:8000‘ in your web-browser and use an instance of the application for development.

Debugging

If you are using this to develop the application, you should use ‘vagrant ssh‘ to login to the machine and run the following commands to figure out what is/is not working:

```
Is xzesd running?
$ sudo systemctl status xzesd
Is apache running?
$ sudo systemctl status apache2
What is the output of xzesd? (logs)
$ sudo journalctl -xe
Print all apache logs as they are generated.
Go to your browser and try using the website.
Only works if Apache is running (above)
$ sudo tail -f /var/log/apache2/*.log
```

About the setup script

The setup script in this directory is used to directly provision a stock Debian box.
If you intend to deploy this application, start with that script and work from there.
If you are on a Debian system and have checked out this code repository, you should be able to use the ‘setup.sh‘ script with limited changes to install the application from source.

Variables

The following scripts should be edited in the ‘setup.sh‘ script to correspond with the appropriate change in behavior.

```bash
# Ideally used in automation...
${TRAVIS_BUILD_DIR:="/xzes"}
# Specifies the directory where the repository is checked out
${XZES_SRC_DIR:=$TRAVIS_BUILD_DIR}
# Specifies where to install the application to
```

```
XZES_BIN="/usr/local/bin/"
# Specifies where to push cgi scripts
XZES_WWW_BIN="/var/www/cgi-bin"
# Where to put html/css/js files
XZES_WWW="/var/www/xzes"
# If set to true, copies files instead of sym-linking them
XZES_INSTALL=false
# set this to false if you do not want a symlink to the code in your homedirectory
# otherwise set it to your user name
DEV_USER="vagrant"
```

```
The most important would be `XZES_INSTALL` which is a boolean to either install or not install the
    application, and `XZES_SRC_DIR` which specifices where the checkout of XZES40-Transformer is.

[vbox]: https://www.virtualbox.org/wiki/Downloads
[vagrant]: https://www.vagrantup.com/docs/getting-started/
```

Listing 21: Vagrant development environment README.

10 LEARNING: TECHNICAL

The following subsections outline material and individuals who helped us with technical tasks.

10.1 C/C++ Technologies

In writing C/C++ we used many technologies and read a *lot* of documentation.

For the Xerces and Xalan libraries (used for XML transformation) we used the online documentation (http://xml.apache.org/xalan-c/, https://xerces.apache.org/xerces-c/) as well as examples included in the source tree of those projects as reference when writing our own code that used these libraries. Examples were the main source of truth and proved very helpful.

10.2 Tex Technologies

In writing documentation we used LaTex technologies and Markdown language. The main resource for the LaTex is that we learn from the shareLaTex (https://www.sharelatex.com/project), and we learned the Markdown language from the GitHub documentation page.

10.3 API technologies

The web API used Apache HTTPD and the Python CGI library. The Apache documentation (https://httpd.apache.org/docs/) and Python CGI library documentation (https://docs.python.org/2/library/cgi.html) were the main sources documentation used for crafting this task.

10.4 Web technologies

In developing the website a lot of Javascript and jQuery was used. The main source of information for jQuery was the project documentation, found at http://api.jquery.com/.

10.5 Platform technologies

In building an application for Linux there were a few notable learning curves we did not expect.

For development we use Vagrant. This tool was useful but also caused some headaches early on with strange errors to do with mounting directories into the development VM. [1] The vagrant documentation (vagrantup.com) and online reference material (stackoverflow.com) were very helpful in debugging these problems.

In developing for Linux Elijah created a systemd service so that the application would be managed, and more importantly monitored and rebooted, by the Linux service manager. Systemd units are not the easiest thing in the world to write, but the systemd documentation was very helpful. This can be found online at
https://www.freedesktop.org/wiki/Software/systemd/.

11 Learning: Personal

11.1 Zixun Lu

I have learned lots of knowledges from this project. In technical knowledge, this project has lots of basic programing skill train, so I am a better C/C++ programmer. I was learning C/C++ in my first year of university life, and then I take python and other program languages to finish my work. This project pull me back to the C/C++ programing, and I never finish this so big project before. It's struggle at the beginning, but actually I get used to it during I write a code, and I review my C/C++ programing knowledge before I start write true code. The most important technical knowledge that I have learned is multiple threads and memory manager. The first year teaching only tech you the basic idea about the C++, but it is not enough for true program. We usually want have higher speed with our program, so I learned how to do it. I used the Pthread library which is the POXIS strander to speed up my program. Because we have the 1. We wanted to mount the directory in the VM so we could simply spin up the development VM, write code on our computers with our favorite IDEs, and just use the VM to build and test without requiring constant re-provisioning or manual copy/pasting of the code into the VM. 80 in-memory cache for our project, I have learned the C++ memory manager problem. Basically, I am responsible for the back-end system implementation, and I come to Elijah to integrate back-end to the front-end. In non-technical knowledge, I have learned how to track the progress of the project, how to manager a project, and how to communicate with teammate and client. The progress tracking is important, because it 9 month project, it is impossible we remember everything in our mind. The Capstone project is my first big project in my university life, so I take time to learn how to manage project. Elijah is kind of experience people, because he was a leader in many project, I learned lot of things from him. First is the time management for a team, we should have frequency contact in a team, and then we should come together to talk about the project before we start writing a program. We should contact the client for clear the requirement early, so it save the time to match our time schdule. The most important things for a team is that we should keep us in a same page, so none of us confused and did somethings worry. For the teamwork, I was lucky before. Everyone did the fair job, so we finished our project early. However, this project is not. I understand that it hard to said I don't know about this technologies for a student, but I don't understand why computer science student dose not learn a technologies. It looks like that he will say that I will finish my work soon, but actually he never did anything. Thus, we have to re-assign our work, and because our project contain lots

1. We wanted to mount the directory in the VM so we could simply spin up the development VM, write code on our computers with our favorite IDEs, and just use the VM to build and test without requiring constant re-provisioning or manual copy/pasting of the code into the VM.

of lower-level programing skills, one of our teammate just did nothing. In this situation, I have leaned if one of our teammate is lack of technical knowledge, and if we can't fire it, we should just give him/her a simple job such as writing documentation or something easy to do. If I can do my project all over again. I may did the same things as before, but I should have little tips for myself. • It is better that have a prototype for a big project, because it gives the initial idea how this program works, and then we improve it to meet our requirement. • Do not rely on your teammate, even if your teammate is good person, but you should do your work by yourself. If your teammate is lazy person, and you have to do your work by yourself too. • I should check the requirement even after I finish my work. I should ask myself that is that good? is that enough? and then I should talk with client to check if it is good or not.

11.2 Shuai Peng

I have learned lots of knowledges from this project. In technical knowledge, this project has lots of basic programing skill train, so I am a better C/C++ programmer. I was learning C/C++ in my first year of university life, and then I take python and other program languages to finish my work. This project pull me back to the C/C++ programing, and I never finish this so big project before. It's struggle at the beginning, but actually I get used to it during I write a code, and I review my C/C++ programing knowledge before I start write true code. The most important technical knowledge that I have learned is multiple threads and memory manager. The first year teaching only tech you the basic idea about the C++, but it is not enough for true program. We usually want have higher speed with our program, so I learned how to do it. I used the Pthread library which is the POXIS strander to speed up my program. Because we have the in-memory cache for our project, I have learned the C++ memory manager problem. Basically, I am responsible for the back-end system implementation, and I come to Elijah to integrate back-end to the front-end.

In non-technical knowledge, I have learned how to track the progress of the project, how to manager a project, and how to communicate with teammate and client. The progress tracking is important, because it 9 month project, it is impossible we remember everything in our mind. The Capstone project is my first big project in my university life, so I take time to learn how to manage project. Elijah is kind of experience people, because he was a leader in many project, I learned lot of things from him. First is the time management for a team, we should have frequency contact in a team, and then we should come together to talk about the project before we start writing a program. We should contact the client for clear the requirement early, so it save the time to match our time schdule. The most important things for a team is that we should keep us in a same page, so none of us confused and did somethings worry. For the teamwork, I was lucky before. Everyone did the fair job, so we finished our project early. However, this project is not. I understand that it hard to said I don't know about this technologies for a student, but I don't understand why computer science student dose not learn a technologies. It looks like that he will say that I will finish my work soon, but actually he never did anything. Thus, we have to re-assign our work, and because our project contain lots of lower-level programing skills, one of our teammate just did nothing. In this situation, I have leaned if one of our teammate is lack of technical knowledge, and if we can't fire it, we should just give him/her a simple job such as writing documentation or something easy to do.

If I can do my project all over again. I may did the same things as before, but I should have little tips for myself.

- It is better that have a prototype for a big project, because it gives the initial idea how this program works, and then we improve it to meet our requirement.
- Do not rely on your teammate, even if your teammate is good person, but you should do your work by yourself. If your teammate is lazy person, and you have to do your work by yourself too.

- I should check the requirement even after I finish my work. I should ask myself that is that good? is that enough? and then I should talk with client to check if it is good or not.

11.3 Elijah C. Voigt

I tried early on in this project *not* to be the leader of the group. I have been the defacto leader of many groups and it is rarely rewarding and is *always* time consuming. This project was similarly time consuming and not very rewarding, but I did learn a lot of useful lessons along the way.

When someone on your team does not have the technical skills to complete a task, they will not tell you this until you build some level of trust and actually test them. Once you've built trust with somebody they will tell you "I do not have the technical skill to do this." Before you've built trust there is shame in revealing this, even if you *do not care about their shame*; you just want something to get done as quickly as possible, but they will struggle to get it done until you say "have you done that thing yet?" and they will keep saying "Not yet, but I am closer." It's stressful to say you don't know how to do something, going forward I hope to more easily recognize when I am not ready to complete a task knowing now that this is exactly what my manager needs to know.

Once you've established that someone on your team isn't technically proficient enough to complete a task you need to figure out what to do next. You should (probably) reassign tasks immediately so that everybody is doing what they're capable of. In addition to this you will probably need to spend one-on-one time with struggling developers to help them get up to speed on their new tasks and to make sure they *actually can* do their new task, otherwise you'll need to re-assign tasks later.

Last I've learned that projects are not designed to be fair. You're not going to be assigned to a five person project where you do exactly one-fifth of the work. You might to a little more, or a little less, or even a *lot* more, that's just how it goes. It doesn't mean that live hates you, it just means you're on a team of people with diverse backgrounds and proficiencies. You should voice your concerns to your boss if something feels *really* out of whack, but don't take it personally if the person you're working with isn't doing their one-fifth. Just don't.

I learned a lot of useful technologies, some of which I had a conversational understanding of but had never really dived into myself. These mostly included Javascript, jQuery, Apache CGI scripting, Linux's systemd, and I got a lot better at writing (read: debugging) C and C++ code.

If I could do this project again I would make the following changes:

- **I would not write skeleton code for the entire project.** This was a good idea at the time, but a lot of the code I wrote for this was thrown out because of weird bugs we couldn't solve. Instead if we had focused on getting *something out the door* as soon as possible we would have had something to show our client earlier in development and we would have ultimately produced a better product which was more feature rich. The product we ended up completing was not functional until every piece had been completed, which took a lot of time and was very difficult to execute on. Sometimes you've just got to be a little scrappy to get your project done.
- **I would stick to my lane.** I ended up picking up the slack of one of my teammates which caused me a lot of undue stress and wasn't much fun for him either. If I had just done what I was responsible for I would have gotten my part of the project done and would have been a little less stressed. On the other hand a lot of the project probably would not have gotten done, and the reason I picked up the slack was not because I wanted to do more than my share of work but because I wanted the project to be good. Ultimately it's my own damn fault, but it is still something I would try to do differently in future projects.

- **I would have worked to get a more complete requirements list.** A lot of the project's requirements either shifted or grew more defined the further into development we went, this caused some headaches. For example, parameter and additional file passing was not something we new we needed to add until the end of winter term. I asked our client and teammates if we knew that we were meeting our requirement needs, to which everybody said yes, but the question was not quite asked correctly. Going back to point one, if we had a demo to show our client we could ask "Is this good enough?" and get feedback about additional features we needed instead of blindly walking toward our deadline just to realize (on my own) that we were missing features which were necessary for end users and to benchmark the application. This was frustrating more than anything, but honestly if we had just stuck to our original (not really good enough for production) requirements I doubt anybody would care; this is just my personal angst about making something that people will actually use instead of spending a year on a prototype that will be put away in a dusty box.

12 EXPO POSTER

A color copy of the poster presented at expo can be found on the following page.

Why we made XZES40 Transformer.

Useful, Challenging, and Open Source

This project was chosen because it provided students the opportunity to create a real-world impact, solve an interesting problem, and contribute to the Free, Libre, and Open Source Community (FLOSS).

The software may be used by individual, organization, and enterprise users to XML perform document transformations. This is a necessary and time consuming task which can now be done faster and *in the cloud*. It proved technically challenging in that it required an understanding of C programming, in-memory caching, parallel computation, application networking, and extensive library usage. The final product was shared with the FLOSS community, giving both our client, the Apache Software Foundation, and any the rest of the world a tool which is useful, usable, and free (*in more than one way*).

Applications and uses

XML is a machine readable data markup language used to store any well-formed topic. XML Stylesheets are used to express transformations of a given XML document. These are used together to store large amounts of data, and "ask" the data a question like "How many employees in this XML file took less than three days of vacation, and make more than $50,000 per year?"

XML doument transformation is used for largely utilitarian purposes in a wide veriety of fields. Police departments, corporate offices, academic institutions, and countless other organizations need and use XML documents and transformations.

The XZES40 application provides an easy to use and access platform for transforming such documents over the web.

CS CAPSTONE PROJECT: XZES40 TRANSFORMER

High Performance XML/XSLT Transformation Server

Optimized for Speed and Convenience

The XZES40 Transformer takes two files as input, an XML document and XSLT stylesheet, and a variable number of custom build parameters or dependency files, and generates a new XML document. While this type of application is not new the concept has been improved with key optimizations and made accessable over the internet via a Web interface.

The strengths of this system are are that it does not need to be installed locally on a user's system, but is still able to perform it's tasks in a timely manner.

The application uses the Apache Xalan and Apache Xerces C++ libraries to perform XML document transformations and a Python CGI script running through an Apache HTTPD server to respond to incomming requests. The transformer daemon is optimized to perform fast transformations by caching previously encountered XML documents and performing document transformations in parallel when possible. The Python CGI script communicates with the Daemon over a local network socket; multiple requests are sent to the daemon and each one is serviced in a POSIX thread. Once the transformation is complete it is sent back to the user.

The web service runs on a remote server atop existing techologies so it can be run, managed, and modified by anybody with Linux and Apache HTTPD experience.

Conclusions/Outcomes

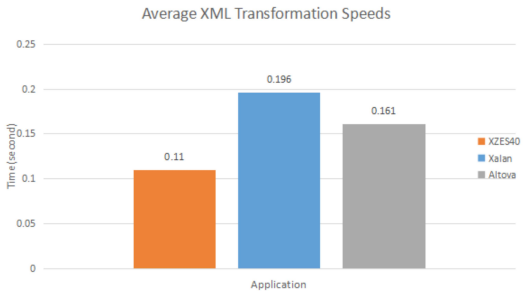
The developed application can generate a new XML file given an XML input, Stylesheet input, and custom build parameters via a Web interface, all without error.

The minimium requirements were completed, like transforming XML documents, as well as many of the optimizations and benefits we set out to complete. The most directl benefit of these being a web interface to access the transformer over the internet. In addition to this we successfully completed a simple in-memory caching system and parallel transformation of the documents. In the end we gave users an application fast enough to be competitive with enterprise software, while still being convenient to usable via the browser.

Results

The following is a table which represents the benchmark results of transforming 19 files 3 times each on the Altova enterprise XML transformer, Xalan Open Soruce transformer, and our XZES40 web Transformer. The files tested came from the Xalan project source code repository.

filename	XZES40	Xalan	Altova
builddocs.xml	0.109s	0.204s	0.187s
buildlibs.xml	0.109s	0.214s	0.193s
charter.xml	0.110s	0.221s	0.190s
ommandline.xml	0.110s	0.201s	0.178s
download.xml	0.113s	0.205s	0.169s
extensions.xml	0.108s	0.196s	0.177s
faq.xml	0.108s	0.188s	0.165s
getstarted.xml	0.108s	0.193s	0.162s
index.xml	0.107s	0.189s	0.151s
Install_save.xml	0.108s	0.212s	0.162s
install.xml	0.110s	0.208s	0.147s
overview.xml	0.110s	0.179s	0.155s
programming.xml	0.108s	0.169s	0.143s
samples.xml	0.106s	0.177s	0.135s
secureweb.xml	0.105s	0.199s	0.139s
test-faqs.xml	0.109s	0.185s	0.136s
usagepatterns.xml	0.121s	0.205s	0.147s
average	0.110s	0.196s	0.161s



The team behind XZES40 Transformer.

Sponsor

Steven Hathaway
The Apache Software Foundation
Email: shathaway@apache.org

Team Members

Elijah C. Voigt
Email: voigte@oregonstate.edu

Shuai Peng
Email: pengs@oregonstate.edu

Zixun Lu
Email: luzi@oregonstate.edu

About the Team

Elijah, Shuai, and Zixun are are fourth year Computer Science students at Oregon State University. Elijah is an Applied CS student focusing on mathematics and computer security, hoping for a career in systems engineering and games development. Shuai is a Systems CS student and is passionate about understanding and designing complex systems, looking for a career in software engineering. Zixun is a double major in CS and Business, intending to be an entrepreneur.

The team also thanks and appreciates Steven Hathaway who gave much of his time and patience to the project.

13 Appendix 1

14 Appendix 2