Design and Architecture

As a Boilermaker pursuing academic excellence, we pledge to be honest and true in all that we do. Accountable together – We are Purdue.

(On group submissions, have each team member type their name).

Type your names: Ashwin Senthilkumar, Katie Roberts, Parker Bushey

Write today's date: 9/18/21

Assignment Goal

The goal of this exercise is to analyze the design and architecture of prominent acts of software engineering, and learn from the experiences of their engineers. Think of this assignment as taking a virtual tour of the Hoover Dam and a Boeing 747.

Relevant Course Outcomes

This assignment is associated with the following Learning Outcome.

- ii: The ability to conduct key elements of the software engineering process, including
    - designing, including through object-oriented design and the unified modeling language (UML).

Resources

The articles about each project come from The Architecture of Open Source Applications Volumes 1 and 2.

Assignment

In this assignment, your group will consider six articles about the design and architecture of open-source projects. Your group will read these articles and ponder the “what” and “why” behind their design decisions.

Part 1: Design Analysis (divided among group members)

Amongst your group, have each person analyze ***two*** of the following articles.

- LLVM
- FreeRTOS
- Berkeley DB
- The Bourne-Again Shell (Bash)
- nginx
- Matplotlib
- Twisted
- The Hadoop Distributed File System
- Audacity
- Moodle

For each article, a team member should answer these questions:

- What is the purpose of the project? (What need does the project fulfill?)
- Whom is the project intended to help? (Who are the users?)
- Describe the design, part 1: High-level.
    - What data does the system process?
    - What computations are performed on this data?
    - How does the system coordinate the data and computation?
- Describe the design, part 2: Modular decomposition.
    - What are the major components?
    - How do these components interact?
- Analyze the design: Flexibility/Resilience to change.
    - Give an example of a change in requirements that this design could easily accommodate. For example, a changing (but predictable) requirement would be isolated to one or a small number of modules.
    - Give an example of a change in requirements to which this design would ***not*** be resilient.

A template follows.

> *On academic honesty: You are welcome to re-use images from the articles, but prose should be your own.*

In Part 2, you will discuss as a group.

Article 1: Audacity -- Analysis by Parker

What is the purpose of the project? (What need does the project fulfill?) (*Write a paragraph)*

Audacity has an outdated purpose, and a more modern purpose that it gained after becoming open source software. When it was created, the software was intended to be a debug tool for audio processing algorithms. Nowadays, people know it as a tool that can record and edit audio. The need that Audacity fills is that of an audio editing software that is easily accessible for new users.

Whom is the project intended to help? (Who are the users?) (*Write a paragraph)*

Due to the accessible nature of the software, Audacity is targeted at an amateur audience of people that work with audio files. However, Audacity is not just simple to use. Built into the software are many other "optional" features that add lots of helpful features. These features are not useful to the nonprofessional user, so Audacity has an additional user base of more experienced individuals. The inclusion of these additional tools makes the program useful to a wide range of proficiency levels among its users.

Describe the design, part 1:

- What *data* does the system process?
- What *computations* are performed on this data?
- How does the system coordinate the data and computation?

Audacity processes sound data from a variety of file formats such as MP3, FLAC, OGG, and Vorbis. However, the latter options are only available through optional plugins due to licensing restrictions.

Audio that is loaded into Audacity can be trimmed or have sections cut out of it. This is done using what they call BlockFiles. Each audio file is split up into small, manageable chunks with their own file format (AUP). Using this system, insertions and deletions are easy even for very large audio files.

Effects can be added to audio files, but not in real time. Due to the nature of the software, it is likely to be used on all sorts of machines that are fast or slow. To ensure a smooth experience, effects are rendered first before allowing playback. This way, even if Audacity is running on a very slow computer, there won't be any issues with audio effects playing.

A more general idea for the types of computations that can be performed on audio files has to do with scripting. While not a strict computation type, Audacity does allow plugins that support scripting languages to make modifications to audio. A user can write code that the plugin delivers to Audacity and controls the software programmatically.

Describe the design, part 2:

- What are the major components?
- How do these components interact?

The major components of Audacity are the two GUI libraries, PortAudio, TrackPanel, BlockFiles, and effects.

The main GUI library is called wxWidgets and it provides all of the front end features like buttons and popups. Just underneath that layer of GUI is a component called ShuttleGUI whose only job is to ferry information to wxWidgets in a way that involves significantly fewer lines of code. These two pieces interact directly to make the software less error-prone by making GUI calls simpler to read and write.

Next is PortAudio which allows audio recording and playback in Audacity. This library allows this functionality regardless of what system Audacity is running on. It provides abstraction to the process of utilizing a system's audio card which is quite different depending on the operating system. This allows PortAudio to interact directly with the computer's ability to play audio.

The TrackPanel is what the user interacts with to make edits to the loaded audio file. The TrackPanel utilizes wxWidgets to display the information needed. Other things such as position on the screen is taken care of by code written specifically for Audacity. The author mentions that the code for this module is horribly organized, but such things arise when managing open-source software.

BlockFiles are what Audacity uses to load audio files into a format that is editable by the user. The audio playback interacts with this module for obvious reasons, but in addition to that the TrackPanel module uses BlockFiles as well. A user can zoom in and out of the timeline for an audio file and the BlockFiles dynamically change to provide different levels of resolution for the GUI parts of Audacity.

The effects built into Audacity work alongside the BlockFiles to change the audio waveforms. In addition to the BlockFiles, this module also has an effect on the GUI as Audacity does not allow for real time playback of added effects. The GUI is unresponsive while an effect is being applied, so the GUI must know when to not accept user inputs.

Analyze the design: Flexibility/Resilience to change.

- Give an example of a change in requirements that this design could easily accommodate. For example, a changing (but predictable) requirement would be isolated to one or a small number of modules.
- Give an example of a change in requirements to which this design would ***not*** be resilient.

The addition of a new supported scripting language would be a requirement that the design of Audacity could handle effectively. This is because scripting is something that the inner working of Audacity have no view of. A separate module takes the outside scripting language and turns it into instructions for Audacity to complete. This encapsulation of the scripting means that if something were to go wrong, it would most likely be from the scripting language and not Audacity itself.

A change that would cause problems for Audacity would be a change to the TrackPanel. It is mentioned in the article that the code for the GUI elements and the Audacity specific elements are not well separated. A change to one part of this would require working in both the Audacity side and the GUI side because of their inefficient mingling in the code. This would likely result in unforeseen consequences to parts of code that didn't need to be changed should the requirements evolve.

Article 2: Moodle -- Analysis by Parker

What is the purpose of the project? (What need does the project fulfill?) (*Write a paragraph)*

The classroom is just one example of a medium for school activities to take place. Sometimes it might be more convenient to interact on the internet to allow more flexibility for all parties involved. Moodle is one such tool that allows this type of learning. It acts as a virtual classroom that can be used to give lessons and distribute materials related to a course.

Whom is the project intended to help? (Who are the users?) (*Write a paragraph)*

Moodle is aimed at helping teachers and students. A teacher's goal is to teach a course and give student materials that will help them succeed. A student's goal is to learn the materials provided for the course. These two groups have a need to interact in order to achieve their goals, so Moodle allows communication in a structured way between them.

Describe the design, part 1:

- What *data* does the system process?
- What *computations* are performed on this data?
- How does the system coordinate the data and computation?

Moodle deals with two main types of data. The first being a database and the second being any number of files and file types. For database calls, a number of different databases will work including MySQL or Microsoft SQL server. For the file management side of Moodle, files need to be able to be uploaded. These two aspects of Moodle are coordinated over the web server that it's hosted on and both need to be readable and writable from it.

Describe the design, part 2:

- What are the major components?
- How do these components interact?

Moodle has three main built-in components which are the actual app code, the database, and the folder for file storage. These components all interact with each other on a web server. If multiple servers are being used, the app code can be copied over multiple servers, while the database and file storage would be on one.

In addition to the main three components that Moodle comes with, the program is also compatible with a number of outside programs called plugins. The plugins are used by Moodle through an API provided by the plugin. The plugins are strongly typed so each plugin only provides a certain type of functionality. This allows Moodle to add additional functionality while keeping the code for such functionality outside of the Moodle core.

According to the article, Moodle is accessed using URLs. To make sure not anyone with a URL can access whatever they want, there is an authentication system in place that is on top of everything else. Certain roles have certain permission and logging into the system allows access to what these roles should have access to.

Analyze the design: Flexibility/Resilience to change.

- Give an example of a change in requirements that this design could easily accommodate. For example, a changing (but predictable) requirement would be isolated to one or a small number of modules.
- Give an example of a change in requirements to which this design would ***not*** be resilient.

Anything having to do with an addition of a new plugin to its list of supported plugins should be simple with the given layout. Since each plugin operates outside of Moodle through an API, if anything were to go wrong while using the plugin, the plugin itself would most likely be the culprit.

Something more difficult to make changes to would likely be changes to the types of data needed in the main code of Moodle. If a new piece of information is needed by the main code, the structure of the database would have to be changed as well. If the database is not managed carefully, it could cause other problems in the main code.

Article 3: Twisted -- Analysis by Kaitlyn Roberts

What is the purpose of the project? (What need does the project fulfill?) (*Write a paragraph)*

Twisted is for developing games and networked applications in a scalable, cross platform framework in Python. At the time of development, there were no cross-platform libraries for game developing, so there was a clear need for a framework that could help run event-driven games and networked apps on multiple different devices easily, without needing to rewrite the code to comply with a new network protocol all the time. Twisted is an engine that manages network connections between clients and servers by allowing objects to make event-driven networked method calls.

Whom is the project intended to help? (Who are the users?) (*Write a paragraph)*

Game developers and network application writers looking to run network events via an Object method call.

Describe the design, part 1: high level

- **What *data* does the system process?**
- **What *computations* are performed on this data?**
- **How does the system coordinate the data and computation?**

Twisted handles network events/tasks that asynchronously utilize a single networking thread. This means that the event-driven tasks using the thread do not have to wait till the one before finishes to make network calls on the thread. When the current task is not using the thread a "callback" is stored, allowing the current task to continue with non-threading code while another task uses the thread, then when the task needs the thread again it puts the callback back into the stack of events waiting on the thread to be available.

Describe the design, part 2: modular

- **What are the major components?**
- **How do these components interact?**

The major components of Twisted are the Reactor, Transports, and Protocols. A Transport is an object that describes a network connection, specified by the Protocol being used. The reactor handles the timing of network events on the thread using a specific Transport and sends callbacks to the calling task when the event completes. The way that the reactor handles and sends events to the thread is determined by the Protocol and API being used.

Analyze the design: Flexibility/Resilience to change.

- **Give an example of a change in requirements that this design could easily accommodate. For example, a changing (but predictable) requirement would be isolated to one or a small number of modules.**
- **Give an example of a change in requirements to which this design would *not* be resilient.**

Changing network protocols could be easily accommodated since the portability of Twisted means that many protocols are already supported and implemented, and can even be used together at once.

Having requirements to manage and monitor networks via administrator privileges has not yet been implemented in Twisted, so adding those requirements would mean the programmers need to rewrite some of Twisted code and add functionality to it. This is a lot of work that may prove to be unreasonable.

Article 4: FreeRTOS -- Analysis by Kaitlyn Roberts

What is the purpose of the project? (What need does the project fulfill?) (*Write a paragraph)*

FreeRTOS is a simple real-time operating system for embedded systems that any programmer can learn to use. It provides needed flexibility and adaptability to an open-source embedded operating system by utilizing global function calls to methods that can change to meet the compiler and device setup.

Whom is the project intended to help? (Who are the users?) (*Write a paragraph)*

Programmers designing embedded systems

Describe the design, part 1: high level

- **What *data* does the system process?**
- **What *computations* are performed on this data?**
- **How does the system coordinate the data and computation?**

FreeRTOS handles User-defined C function tasks stored in task control block (TCB) structs. Tasks can be created, prioritized, scheduled, maintained, and run via FreeRTOS. Tasks and interrupt tasks use the communication queue to send data between tasks or between a task and the hardware interface. The "system tick" interrupt is used to synchronize the tasks with the hardware.

Describe the design, part 2: modular

- **What are the major components?**
- **How do these components interact?**

The major components of FreeRTOS are the communication queue, Tasks, and hardware interfacing. Data/tasks get sent to hardware using global functions defined by whichever "portmacro" file is needed for a particular compiler/processor setup. This makes it so that the tasks only need to call a universal function for common calls, regardless of compiler or processor, rather than the programmer needing to know the setup to make the right function call. All tasks are stored on categorical (circular, linked) lists that specify important metrics about the task, such as what priority it has at the current time and the current state of the task (ready, suspended, running, blocked). Order of tasks on the queue is determined by priority levels assigned to the tasks. The queue is a list of categorical priority level lists of tasks. When different tasks on the queue share a resource, the lower priority task currently using the resource inherits the priority of the other, making sure that resource can be used by the higher priority task faster.

Analyze the design: Flexibility/Resilience to change.

- **Give an example of a change in requirements that this design could easily accommodate. For example, a changing (but predictable) requirement would be isolated to one or a small number of modules.**
- **Give an example of a change in requirements to which this design would *not* be resilient.**

Changes of hardware/processor to run on could be easily accommodated since only the hardware interface portion would be affected (specifically the "portable" header file that includes the correct "portmacro" file), and FreeRTOS supports many compilers and processors.

Changing specs of such hardware could also be accommodated since only the "config" header file would likely be affected.

Changes in the descriptions and specs of tasks the client wants the device to run, since these tasks are user-defined and inherently designed by programmers for very specific embedded uses.

Article 5: Matplotlib -- Analysis by Ashwin

What is the purpose of the project? (What need does the project fulfill?) (*Write a paragraph)*

Matplotlib is a data visualization library for Python that plots/graphs data using built-in packages that makes graphing easy. This project fulfils the needs of anyone using Python for data visualization and plotting graphs directly in the code itself. Since it is in Python, there is more flexibility with being able to connect to external programs and software with more simplification

Whom is the project intended to help? (Who are the users?) (*Write a paragraph)*

Matplotlib is intended for anyone who is doing data visualization in a project and wants to use it for modeling. The graphics that are included in the library, like scatter plots and histograms, help with easy visuals. Matplotlib is used by anyone in the scientific community using Python to visualize graphs or use it as a part of a larger program. It is also used by students and those in data science/statistics who use it to display graphs and charts.

Describe the design, part 1:

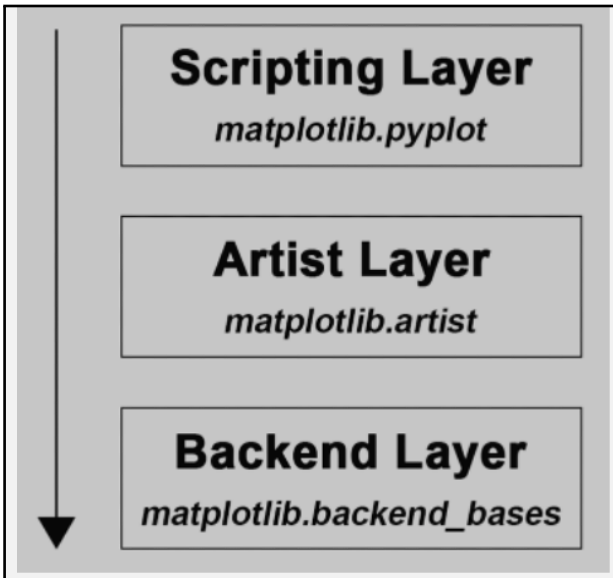
- What *data* does the system process?
- What *computations* are performed on this data?
- How does the system coordinate the data and computation?

Matplotlib mainly takes in numerical data in the form of an array and outputs a Matlab-like graph set. The computations are dependent on what data the program is using to generate a graph. Usually, the data is taken in as columns of a table with the same property and labels on the graphs can be used for axis. Because Matplotlib is a library, the system can coordinate the data and consumption based on what functions the user uses for the input.

Describe the design, part 2:

- What are the major components?
- How do these components interact?

There are 3 key layers, or components, to Matplotlib: Scripting layer, Artist Layer, and Backend Layer. The backend layer handles the heavy lifting by accessing toolkits. The FigureCanvas in this layer is where the graph or figure will get drawn using the Renderer object. The Artist layer is where the user can control different elements of the figure (like an artist and a canvas). This layer is where the most customization can happen. The final layer is the Scripting layer, where the script makes the matplotlib work like a MATLAB script. This is where you can use the command line to make functions that generate the graphs.



Analyze the design: Flexibility/Resilience to change.

- Give an example of a change in requirements that this design could easily accommodate. For example, a changing (but predictable) requirement would be isolated to one or a small number of modules.
- Give an example of a change in requirements to which this design would ***not*** be resilient.

Because matplotlib is considered a very flexible program since it can be used on any python application that imports the library for it, matplotlib is versatile in taking in most data types into it and plotting it correctly. It can also be transferred from one system to another, which makes it universal on anything that can run python. One example of where matplotlib might have some difficulty would be when the input data is taken in in the wrong format. If the formatting of the data is strict and cannot be changed, then we cannot use matplotlib with an incorrect format. If they can make its design in a way that can take in data with differing formats, that will allow matplotlib to be more accessible and have less stricter requirements.

Article 6: Bash -- Analysis by Ashwin

What is the purpose of the project? (What need does the project fulfill?) (*Write a paragraph)*

Bash is a command language interpreter that executes commands entered with a command prompt that processes text files as input. The name is an acronym for Bourne-Again Shell. The project's purpose is to allow anyone to run commands on any system, making this universal for most operating systems. The purpose of bash is to help perform tasks efficiently on any machine. Bash is run on Unix but can also be used for Windows OS.

Whom is the project intended to help? (Who are the users?) (*Write a paragraph)*

Bash's command protocol helps programmers run tests or access file properties without much complication or extra packages. The simplicity of it also allows for users of various programming backgrounds and levels of expertise to run bash commands easily. The interactive debugging feature makes testing code easier as well.

Describe the design, part 1:

- What *data* does the system process?
- What *computations* are performed on this data?
- How does the system coordinate the data and computation?

Bash commands are run on the terminal and can retrieve data from specified files and run them with installed packages. Once entered, the command should ideally perform the task and return an output on the screen. As far as computations go, it can use a shell script specified in a file that will read it and execute its commands. Once these commands get executed, the operations and data manipulation will occur based on what's specified in the command line. Essentially, the shell action invokes an executable and causes the kernel to create a new running process.

Describe the design, part 2:

- What are the major components?
- How do these components interact?

The major components of Bash includes "input processing, parsing, the various word expansions and other command processing, and command execution, from the pipeline perspective. These components act as a pipeline for data read from the keyboard or from a file, turning it into an executed command" (Chet Ramey, Linked Article). The input processing takes characters from the terminal line or the linked file and uses the shell parser to parse through each line and run these lines as commands. The shell variables are treated as strings. This is where we use the example of argv that contains the input arguments of a file or program.

Analyze the design: Flexibility/Resilience to change.

- Give an example of a change in requirements that this design could easily accommodate. For example, a changing (but predictable) requirement would be isolated to one or a small number of modules.
- Give an example of a change in requirements to which this design would ***not*** be resilient.

One of the main benefits of Bash is that it's a flexible interface and can be run on different environments and operating systems. The word expansion part of its components allows for it to work with the command line when the user can put in their specific command easily. One example of where a change in requirements that this design would not be resilient with would be if the user does not enter the command in the specific format that bash requires, so it would throw an error rather than take in a different format of input.

Part 2: Synthesis (group activity)

Now that your group has read independently about designs and architectures, let's take some time to synthesize and apply what you've learned.

Manifestations of stages of the engineering process

In class we've discussed the five fundamental stages of the engineering process: Requirements, Design (& Tradeoffs), Implementation, Validation, and Maintenance. To what extent did your team observe these stages manifesting in the articles? Give an example of each stage, taken from any of the articles you read.

Example of requirements:

- Audacity
- The team working on Audacity has a very clear requirement that is the core of their design for the software. They state that the software should be easily learnable and come ready to use without need of a manual. This requirement is vague, but it guides a lot of their decisions when it comes to user interface and base functionality. The user interface is clear and understandable, and the functionality not too extensive as to overwhelm an ameteur. Additional functionality can be added on by those who want it, but it isn't knowledge that is required to use the software.

Example of design tradeoff:

- Matplotlib
- One of the main features of Matplotlib is its built-in functions that uses MATLAB APIs to synthesize the information and allow the library to make plots/tables based on the data inputted. It also meant that those who are already familiar with MATLAB will have a much smaller learning curve using matplotlib compared to completely new users. One design flaw that the author points out with it is the library's lack of integration early-on with pre-existing python APIs.

Example of Implementation:

- Moodle
- The development of Moodle began in 1999 and finally released in 2002. The requirements had likely been laid out beforehand. Generally speaking, the requirements would be for teachers and students to be able to communicate and for class materials to be distributed. Over this almost three year period, the developer Martin Dougiamas was working on implementing his design ideas.

Example of Validation:

- Twisted
- Since twisted is a network communication engine, it has to conform to certain standards, such as RFC. Robust tests were implemented extensively and frequently to validate the engine's conformance to those standards.

Example of Maintenance:

- Twisted
- After the initial release of Twisted, the team continued to work on improvements to the system, such as a class called Deferred. Other improvements were in the works for years, however these projects either introduced unneeded complexity or were too ambitious for the team to take on. For example, the team tried to rewrite a package called web from the ground up. Web2 was accidentally released unfinished and therefore confused users. The author states that if the team were to try implementing web2 again, they would have done so in incremental updates instead of side by side with the original web package.

Dwelling on lessons learned

Each engineering project article concluded with some "lessons learned" / "reflections". Discuss them amongst yourselves. Indicate two lessons that your team found particularly poignant. What struck you about these lessons – something surprising, something insightful, something fresh?

Lesson 1: *When implementing your design, don't be afraid to start with the simplest idea instead of making it too complex.*

- We think that this is a particularly helpful idea because as of milestone 1, we had this same issue. We were trying to do something simple in a way that we were informed was more complex than it needed to be. Luckily we got some advice early on before trapping ourselves with our initial design. We think beginning with a simple design and then adding features as needed is a good alternative to starting complex. If you start too complex, you can lose sight of what the original goal of the design was.

Lesson 2: *When designing, make sure it is modular and abstract so that code can look clean and be reused easier*

- Many of the articles we read stressed the importance of modular design and implementing higher levels of abstraction so that the user does not need to understand the deep hard to understand functionalities of your program. Since we are implementing an object oriented Java based design, our abstraction is in the form of our scores package and objects that hold the deeper functions of our code, so that when the user looks at the main code, it is easy to understand even though we are implementing complex API calls in the object functions.

Applications for your project

You've already submitted a project plan for Project 1, including your preliminary design. Based on what you've learned from other engineers in this assignment, would you like to make any changes you might make to your design and/or plan? Discuss. (*If you do make changes, indicate them in your next project milestone document.*)

We included more object oriented parts to the project to create the modularity of each component we will be calculating for.

We plan on explicitly stating scopes of variables and functions so that only the high level abstract code can be seen from the outside.
We plan on converting the project to a Maven project because this would simplify how we implement our dependencies.