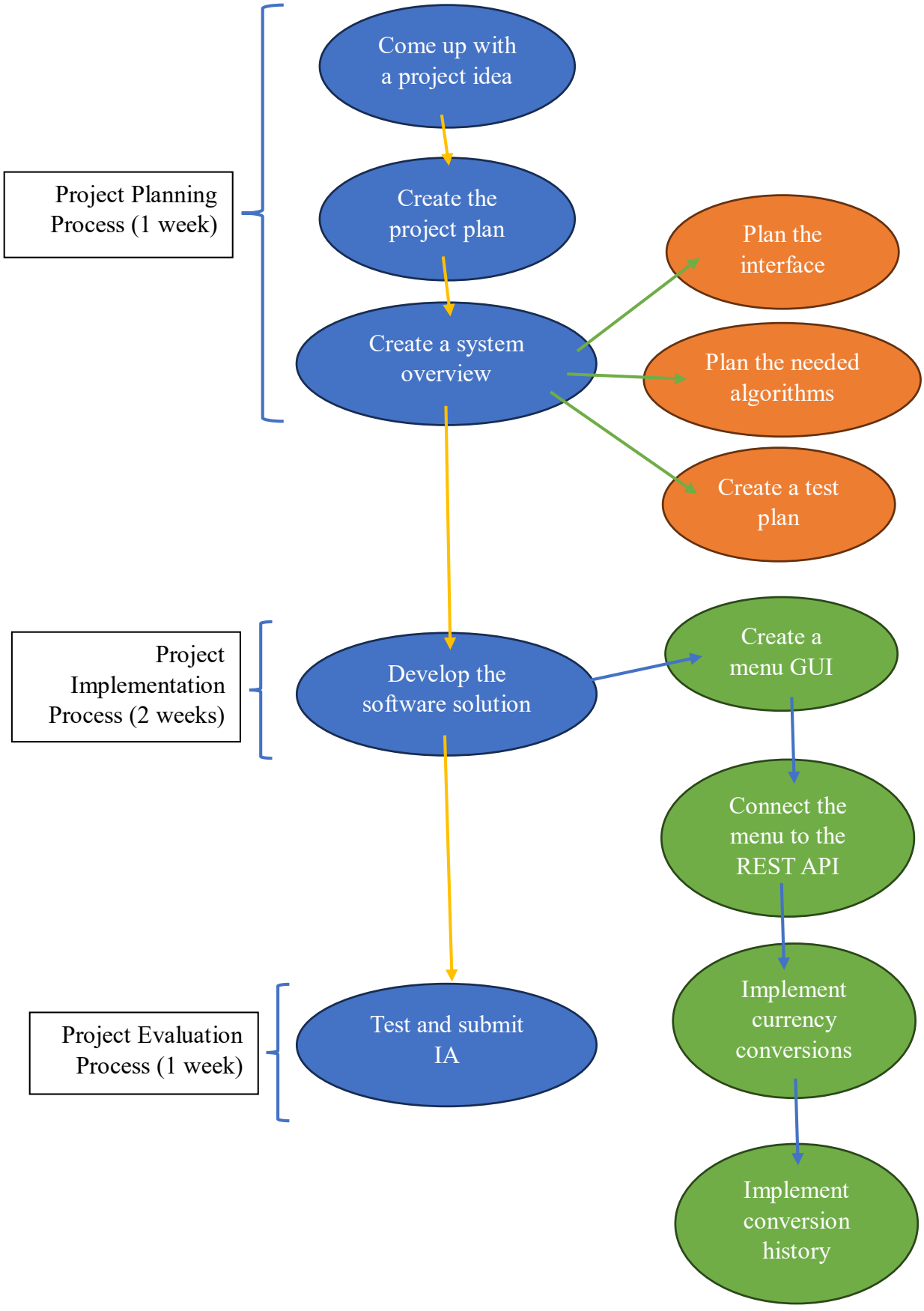
# Criterion A

My father has recently been doing work for an international company in another country, and the payments that he has been receiving were in another currency, one that he does not use often, so he has been struggling with keeping up with the exchange rates and values of the receiving currency, compared to the one he usually uses in our country. I realized that constantly searching for exchange rates will be an inconvenience for me too once I go to another country, so I decided to recognize this as my selected real-world problem, and I decided to create a computational solution that will feature all major currencies and will be able to convert them at any time, as well as save the history of past conversions. This allowed me to declare my success criteria:

1. CurrencyConverter will feature the selection of 2 currencies from a selection of many different worldwide currencies.
2. CurrencyConverter will be able to convert a user-selected amount of one currency to any other currency available.
3. CurrencyConverter will use a real-time REST API to access the current exchange rates of currencies.
4. CurrencyConverter will be able to cache results and rates in order to increase performance.
5. CurrencyConverter will be able to save a history of past conversions between software restarts.

To make this project a reality, I will be using Python for development, including its libraries Kivy (for the development of a simple, yet functional and appealing graphical user interface), DiskCache (to cache results and rates), and Requests (to create HTTP requests to an external REST API to access currencies and exchange rates). The reason for this choice is that it offers all of the possibility to implement all of the features I need, it is my personal preference, and the syntax is very simple to read and understand, and therefore extend and refactor. Additionally, I will be using ExhangeRate-API as my REST API of choice, since it is free, open, and accessible to anyone without the need for an access key, but still providing all of the data that I will need.

# Criterion B


Project Planning Process (1 week)
Come up with a project idea
Create the project plan
Create a system overview
Plan the interface
Plan the needed algorithms
Create a test plan
Project Implementation Process (2 weeks)
Develop the software solution
Create a menu GUI
Connect the menu to the REST API
Implement currency conversions
Implement conversion history
Project Evaluation Process (1 week)
Test and submit IA

# Criterion C

## 1. Algorithms I will use

### 1.1. Access currency exhage rate

```
method get_exchange_rate(base_currency):
    if base_currency exists in cache:
        return cache[base_currency]

    response = send_request_to_API(API_URL + base_currency)
    if response.status_code == 200:
        data = parse_json(response)
        if data['result'] == 'success':
            cache[base_currency] = data['rates']
            return data['rates']

    return None
```

### 1.2. Convert one currency to another

```
method convert(instance):
    try:
        amount = float(instance.amount_input.text)
        from_currency = instance.from_currency.text
        to_currency = instance.to_currency.text

        rates = get_exchange_rate(from_currency)
        if rates is not None:
            result = convert_currency(amount, from_currency, to_currency, rates)
            result_text = concatenate_strings(amount, from_currency, result,
to_currency)
            instance.result_label.text = result_text
            add_to_history(result_text)
            save_history()
        else:
            instance.result_label.text = 'Error fetching exchange rates'

    except ValueError:
        instance.result_label.text = 'Invalid amount'
```

1.3. Load conversion history

```
method load_history():
    try:
        open_file = open(conversion_history_file, 'r')
        conversion_history = read_lines(open_file)
        close_file(open_file)
    except FileNotFoundError: pass
```

1.4. Create app instance

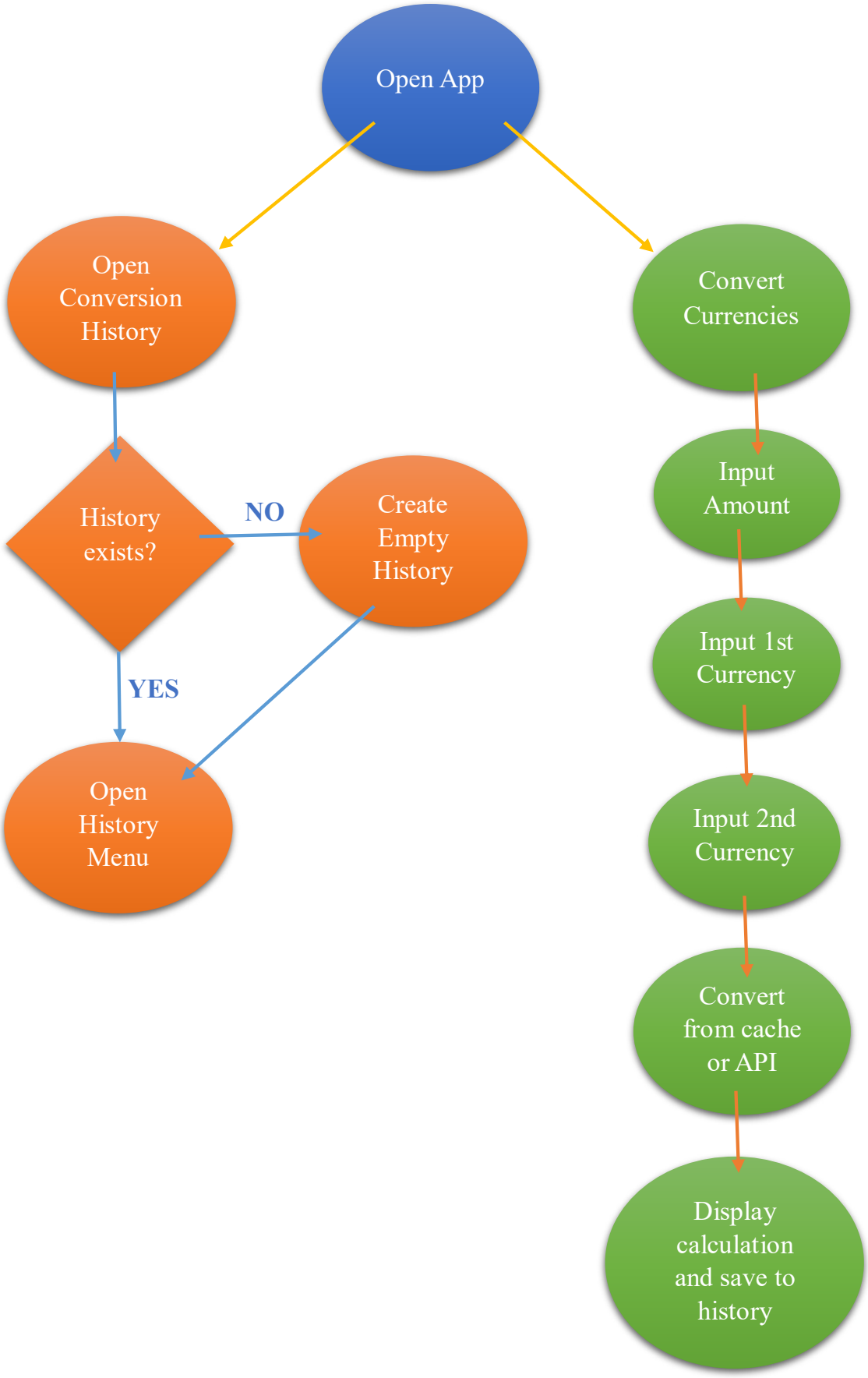
```
class CurrencyConverterApp:
    method build():
        load_history()
        sm = create_screen_manager()
        main_screen = create_main_screen(name='main')
        add_screen(sm, main_screen)
        history_screen = create_history_screen(name='history')
        add_screen(sm, history_screen)
        return sm

    method on_stop():
        save_history()
```

2. Testing plan and strategy

| | |
|---|---|
| **1.1 and 1.2** | Open the REST API and the software one alongside the other. Convert one currency to another and check if it corresponds with what is in the API. Test once more for another currency. |
| **1.3** | Close the application and see if the data is saved in the working directory. Reopen the application and open the conversion history to check if it is persistent. |
| **1.4** | Open the app, expect successful startup.<br>Close the app, expect the history to be saved in a file in the working directory. |

3. System Model



# Criterion D

1. **Saving and loading history of conversions:** This algorithm was implemented in order to be able to persistently store past conversions of currencies even when the software is closed, on the hard drive of the computer. They are straightforward but effective, and they are surrounded by exception handling in order to protect the application from crashing or leaking memory. This algorithm is tested under the testing strategy under 1.4.

```
def save_history():
    with open(conversion_history_file, 'w') as file:
        for conversion in conversion_history:
            file.write(conversion + '\n')


def load_history():
    global conversion_history
    try:
        with open(conversion_history_file, 'r') as file:
            conversion_history = [line.strip() for line in file.readlines()]
    except FileNotFoundError:
        pass
```

2. **Access REST API to retrieve rates and currencies:** In order to obtain exchange rates and offered currencies, CurrencyConverter relies on an external REST API called ExchangeRate-API, in the following algorithm, this API is accessed by sending a request to it and reading the results, which leads to obtaining real-time exchange rates and currencies, exactly what is needed for the software. This is tested under 1.1 and 1.2 in the testing strategy.

```
def get_exchange_rate(base_currency):
    if base_currency in cache:
        return cache[base_currency]

    response = requests.get(API_URL + base_currency)
    if response.status_code == 200:
        data = response.json()
        if data['result'] == 'success':
            cache[base_currency] = data['rates']
            return data['rates']
    return None

def get_available_currencies():
    response = requests.get(API_URL + 'USD')
```

```
        if response.status_code == 200:
            data = response.json()
            if data['result'] == 'success':
                return list(data['rates'].keys())
                 return []
```

3. **Conversion:** The following algorithm is used in order to convert a selected currency to another, which is the main centercourse of the software. It loads all of the data from the GUI fields and passes it as sets of parameters to respective methods in order to complete all calculations and return the results back to the GUI. The testing strategy evaluates this algorithm under 1.2.

```
    def convert(self, instance):
            try:
                amount = float(self.amount_input.text)
                from_currency = self.from_currency.text
                to_currency = self.to_currency.text

                rates = get_exchange_rate(from_currency)
                if rates:
                    result = convert_currency(amount, from_currency,
    to_currency, rates)
                    result_text = f'{amount} {from_currency} = {result:.2f}
    {to_currency}'
                    self.result_label.text = result_text
                    conversion_history.append(result_text)
                    save_history()
                else:
                    self.result_label.text = 'Error fetching exchange rates'
            except ValueError:
                    self.result_label.text = 'Invalid amount'

    def convert_currency(amount, from_currency, to_currency, rates):
        if from_currency == to_currency:
            return amount
            return amount * rates[to_currency]
```

## Criterion E

| Criterion | Evaluation |
|---|---|
| 1 | Two currencies can be selected from a selection of over 150 currencies, therefore, this criterion is successful. |
| 2 | Any amount of any currency can be converted to any other currency. Criterion successful. |
| 3 | ExchangeRate-API is used for free and without the need for an access key. Criterion successful. |
| 4 | Cache is initialized at startup and HTTP request results are cached, which significantly improves calculation performance. Criterion met. |
| 5 | Unless files in the working directory are tampered with, all past conversions will be persistently saved. Criterion mostly successful. |

This software can certainly be improved and extended. A more readable and visually-appealing graphic user inteface would be desirable. Since ExchangeRate-API is rate-limited and is updated only once every 24 hours, a system to allow the user to change their REST API provider would be a great addition to the software. Finally, a good improvement for this application would be to introduce a system to save certain currencies as 'favorites' so that they can be accessed more easily, without the need to constantly search and scroll amongst dozens of currencies to reach the ones that are commonly used.