Code Analysis Report

Analyzed Code:

```
import pickle

def create():
f = open("Library.dat",'wb')
book = {}
print("Enter X to stop")
while True:
bn = input("Enter Name of the Book:")
if bn == 'X'or bn == 'x':
break
bno = int(input("Enter Book no:"))
aut = input("Enter Author of the book: ")
data = [bn,aut]
book[bno] = data
pickle.dump(book,f)
f.close()

def add():
f = open("Library.dat",'rb')
bk = pickle.load(f)
f.close()
n = int(input("No of books to be added:"))
f1 = open("Library.dat",'wb')
for i in range (1,n+1):
bno = int(input("Enter Book no:"))
bn = input("Name of the Book:")
aut = input("Enter Author of the book: ")
data = [bn,aut]
if bno in bk:
print("Book no. already used")
print("Start again")
bk[bno] = data
pickle.dump(bk,f1)
f1.close()
```

```
def delete():
f = open("Library.dat",'rb')
bk = pickle.load(f)
f.close()
nbk = {}
f1 = open("Damaged.dat",'rb')
bno = int(input("Enter book no. of the damaged book:"))
nbk = pickle.load(f1)
f1.close()
f2 = open("Damaged.dat",'wb')
dtd = bk.pop(bno)
nm = dtd[0]
nbk[bno] = dtd
f3 = open("Library.dat",'wb')
pickle.dump(nbk, f2)
pickle.dump(bk,f3)
print("Book with name ",nm ," moved to damaged.")
f2.close()
f3.close()

def edit():
f = open("Library.dat",'rb')
bk = pickle.load(f)
print("What do you want to change?")
print("\t 1. Book name \n\t2.Author Name")
cho = int(input("Enter choice 1 or 2: "))
nbk = {}
bno = int(input("Enter book no. of the book to edit: "))
k = bk.keys()
for i in k:
dt = bk[i]
if i == bno:
if cho == 1:
bn = input("Enter New Name of the Book:")
print("Old name :",dt[0])
dt[0] = bn
nbk[bno] = dt
```

```
elif cho == 2:
aut = input("Enter Author of the book: ")
print("Old name :",dt[1])
dt[1] = aut
nbk[bno] = dt
else:
nbk[i] = dt
f.close()
f1 = open("Library.dat",'wb')
pickle.dump(nbk,f1)
f1.close()

def check_damaged():
f = open("Damaged.dat",'rb')
bk = pickle.load(f)
print("List of Damaged Books")
print("{:<10} {:<35} {:<30}".format('BOOK NO','BOOK NAME','AUTHOR NAME'))
k = bk.keys()
for i in k:
l = bk[i]
print("{:<10} {:<35} {:<30}".format(i,l[0],l[1]))
f.close()

def replace():
f = open("Damaged.dat",'rb')
bk = pickle.load(f)
f.close()
f1 = open("Library.dat",'rb')
nbk = pickle.load(f1)
f1.close()
bno = int(input("Enter book no. of book to be replaced: "))
dtd = bk.pop(bno)
nm = dtd[0]
f2 = open("Damaged.dat",'wb')
pickle.dump(bk,f2)
nbk[bno] = dtd
f3 = open("Library.dat",'wb')
pickle.dump(nbk,f3)
```

```
print("Book with name ",nm ," replaced.")
f2.close()
f3.close()

def borrow():
f = open("Library.dat",'rb')
bk = pickle.load(f)
f.close
nbk = {}
f1 = open("Borrowed.dat",'br')
f2 = open("Library.dat",'wb')
bno = int(input("Enter book no. of the borrowed book:"))
date = input("Enter date of borrowed in DD-MM-YYYY:")
name = input("Enter name of the student:")
nbk = pickle.load(f1)
dtd = bk.pop(bno)
dt = [dtd,name,date]
nbk[bno] = dt
pickle.dump(bk,f2)
f2.close()
f3 = open("Borrowed.dat",'wb')
pickle.dump(nbk,f3)
f1.close()
f3.close()

def return_book():
f = open("Borrowed.dat",'rb')
bk = pickle.load(f)
f.close()
f1 = open("Library.dat",'rb+')
nbk = pickle.load(f1)
f1.close()
bno = int(input("Enter book no. of the book to be returned:"))
Rdate = input("Enter date of return(in DD-MM-YYYY):")
dtd = bk.pop(bno)
dt = dtd[0]
nbk[bno] = dt
f2 = open("Library.dat",'wb')
pickle.dump(nbk,f2)
```

```
f3 = open("Borrowed.dat",'wb')
pickle.dump(bk,f3)
print('{:<35} {:<20} {:<10} {:<10}'.format('Name of Book','Name of
Student','Date Borrowed','Date returned'))
print('{:<35} {:<20} {:<12} {:<12}'.format(dt[0],dtd[1],dtd[2],Rdate))
f2.close()
f3.close()

def books():
f = open("Library.dat",'rb')
st = pickle.load(f)
print('{:<10} {:<35} {:<30}'.format("Book No","Book Name","Author Name"))
k = st.keys()
for i in k:
l = st[i]
print('{:<10} {:<35} {:<30}'.format(i,l[0],l[1]))
f.close()

def check_availability():
f = open("Library.dat",'rb')
bk = pickle.load(f)
bn = input("Enter book name to check availability:")
k = bk.keys()
for a in k:
dt = bk[a]
nm = dt[0]
if nm.casefold() == bn.casefold():
b = 'True'
break
else:
b = 'False'
if b == 'True':
print("Book is available")
print("Book no: ",a)
else:
print("Book not available")
f.close()

def calculate_fine():
b = 10
```

```
d = int(input("Enter no of days: "))
if d <= b:
print("No fine")
elif d > b:
s = (d-b)*b
print("Fine per extra day after 10 days is Rs. 10")
print("Fine amount to be paid is Rs. ",s)

print('\t\t\t\t WELCOME')
print('\tUsers\n \t1. Admin','\n \t2. Student')
user = input("User: ")
if user.lower() == 'admin':
password = input("Password: ")
if password == 'mypass':
print("\t1. Create (*First time use only)\n \t2.Add a book\n \t3.Edit name
or author name:\n \t4.Delete a damaged book\n \t5.Check damaged books\n
\t6.Replace\n \t7.Borrow\n \t8.Return\n \t9.Calculate fine")
ch = int(input("Enter choice 1, 2, 3, 4, 5, 6, 7, 8 or 9: "))
if ch == 1 :
create()
elif ch == 2 :
add()
elif ch == 3 :
edit()
elif ch == 4 :
delete()
elif ch == 5 :
check_damaged()
elif ch == 6 :
replace()
elif ch == 7 :
borrow()
elif ch == 8 :
return_book()
elif ch == 9 :
calculate_fine()
else:
print("Not available")
else:
```

```
print("Incorrect password")
elif user.lower() == 'student':
print("\t1. Books in Library\n \t2.Check Availability")
ch = int(input("Enter choice 1 or 2: "))
if ch == 1 :
books()
elif ch == 2 :
check_availability()
```

Static Code Metrics:

Metric	Value
Lines of Code (LOC)	221
Comment Density	0.0
Cyclomatic Complexity	27
Maintainability Index	26.036570198384652
Halstead Metrics	
Unique Operators	8
Unique Operands	37
Total Operators	28
Total Operands	56
Volume	461.3156600916927
Effort	2792.829942176734
Depth of Inheritance Tree (DIT)	0
Coupling Between Object classes (CBO)	0
Lack of Cohesion of Methods (LCOM)	0
Fan-in	77
Fan-out	77
Number of Methods (NOM)	0

McCall's Intermediate Quality Metrics:

Metric	Value

Modifiability	0.26
Testability	0.36
Reliability	0.48
Understandability	-0.15
Self-Descriptiveness	0.11
Reusability	1.0
Portability	0.68
Efficiency	0

GPT Analysis Report:

Analysis Report on Python Code Improvement

Summary of Issues Identified

The provided Python code is a library management system that manipulates books data using the `pickle` module. Despite its functionality, several areas can be improved to enhance code quality, readability, modularity, and maintainability.

Key Areas for Improvement

1. **Code Organization and Modularity**:

- The code can benefit from splitting functionalities into separate modules or classes. Grouping related functions will help in understanding the flow and will conform to Object-Oriented Programming (OOP) principles.

2. **Error Handling**:

- The code lacks error handling mechanisms. This can lead to crashes or unexpected behavior, especially when dealing with user input or file operations.

3. **File Handling**:

- File handling is performed using raw open/close calls without context managers. Using `with` statements ensures files are properly closed even if exceptions arise.

4. **Input Validation**:

- There is insufficient input validation, especially while converting inputs to integer types. This may cause runtime errors.

5. **Variable Naming and Readability**:

- Variable names like `f`, `f1`, `bk`, `nbk` are not descriptive, making it difficult to understand their purpose without proper context.

6. **Redundant Code**:

- Some code blocks, such as loading and dumping books from files, are repeated across multiple functions. These can be refactored into separate utility functions.

7. **Use of Magic Numbers**:

- The code contains hardcoded values (e.g., fine calculation constant `b = 10`). Consider defining such constants at the top for better readability and maintainability.

8. **Comment Density**:

- The code lacks comments, leading to low understandability. Descriptive comments explaining the purpose of functions and complex code segments should be added.

9. **Testing and Metrics**:

- The code has low maintainability and high cyclomatic complexity (27). This indicates the potential difficulty of testing and maintaining the code due to its complexity.

Suggested Improvements

1. **Refactor into Classes**:

- Consider creating a `Library` class that encapsulates all functionality related to librarian operations, including a `Book` class. This will cleanly separate data from operations.

```
python
class Book:
def __init__(self, book_no, name, author):
self.book_no = book_no
self.name = name
self.author = author

class Library:
def __init__(self, file_name):
self.file_name = file_name
self.books = {}
```

2. **Use of Context Managers**:

- Implement context managers for file operations to ensure proper resource management.

```
python
with open("Library.dat", 'rb') as file:
self.books = pickle.load(file)
```

3. **Error Handling**:

- Introduce `try...except` blocks to catch exceptions during user input and file operations.

```
python
```

```
try:
bno = int(input("Enter Book no:"))
except ValueError:
print("Invalid input. Please enter a number.")
```

4. **Input Validation**:

- Validate user inputs to ensure that they meet expected formats before processing them. This includes checking for empty strings and type mismatches.

5. **Remove Redundant Code**:

- Create reusable functions for common file operations, e.g., loading and saving data.

```
python
def load_data(file_path):
with open(file_path, 'rb') as file:
return pickle.load(file)
def save_data(file_path, data):
with open(file_path, 'wb') as file:
pickle.dump(data, file)
```

6. **Consistent Naming**:

- Use meaningful variable names to improve readability.

```
python
book_data = {}
```

7. **Documentation**:

- Include docstrings for functions and modules to describe their purpose, parameters, and return types. This aids maintainability and readability.

8. **Implement Unit Tests**:

- Develop a set of unit tests that can validate the functionality of each method in the library class. This will improve reliability and make future changes safer.

Conclusion

The suggested changes focus on enhancing the code's maintainability, readability, and functionality. By adopting OOP principles, implementing proper error handling, validating inputs, and ensuring proper resource management, the code can be significantly improved. Additionally, focusing on testing will secure the quality of the application in the long term.