Code Analysis Report

Analyzed Code:

```
import argparse
import datetime
import os
import sys
import time
from argparse import _SubParsersAction
from typing import List, Optional

from db import db
from flask.ctx import AppContext
from management import app_context
from models import Reply, Source, Submission
from rm import secure_delete

def find_disconnected_db_submissions(path: str) -> List[Submission]:
"""
Finds Submission records whose file does not exist.
"""
submissions = db.session.query(Submission).order_by(Submission.id,
Submission.filename).all()

files_in_fs = {}
for directory, subdirs, files in os.walk(path):
for f in files:
files_in_fs[f] = os.path.abspath(os.path.join(directory, f))

return [s for s in submissions if s.filename not in files_in_fs]

def check_for_disconnected_db_submissions(args: argparse.Namespace) -> None:
"""
Check for Submission records whose files are missing.
"""
with app_context():
disconnected = find_disconnected_db_submissions(args.store_dir)
if disconnected:
print(
"There are submissions in the database with no corresponding files. "
'Run "manage.py list-disconnected-db-submissions" for details.'
)
else:
print("No problems were found. All submissions' files are present.")

def list_disconnected_db_submissions(args: argparse.Namespace) -> None:
"""
List the IDs of Submission records whose files are missing.
"""
with app_context():
disconnected_submissions = find_disconnected_db_submissions(args.store_dir)
if disconnected_submissions:
print(
'Run "manage.py delete-disconnected-db-submissions" to delete these
records.',
file=sys.stderr,
)
for s in disconnected_submissions:
print(s.id)

def delete_disconnected_db_submissions(args: argparse.Namespace) -> None:
"""
Delete Submission records whose files are missing.
"""
with app_context():
disconnected_submissions = find_disconnected_db_submissions(args.store_dir)
ids = [s.id for s in disconnected_submissions]

remove = args.force
if not args.force:
remove = input("Enter 'y' to delete all submissions missing files: ") == "y"
if remove:
print(f"Removing submission IDs {ids}...")
db.session.query(Submission).filter(Submission.id.in_(ids)).delete(
synchronize_session="fetch"
)
db.session.commit()
else:
```

```
print("Not removing disconnected submissions in database.")

def find_disconnected_fs_submissions(path: str) -> List[str]:
"""
Finds files in the store that lack a Submission or Reply record.
"""
submissions = Submission.query.order_by(Submission.id,
Submission.filename).all()
files_in_db = {s.filename: True for s in submissions}

replies = Reply.query.order_by(Reply.id, Reply.filename).all()
files_in_db.update({r.filename: True for r in replies})

files_in_fs = {}
for directory, subdirs, files in os.walk(path):
for f in files:
files_in_fs[f] = os.path.abspath(os.path.join(directory, f))

disconnected_files_and_sizes = []
for f, p in files_in_fs.items():
if f not in files_in_db:
filesize = os.stat(p).st_size
disconnected_files_and_sizes.append((p, filesize))

return [file for (file, size) in sorted(disconnected_files_and_sizes,
key=lambda t: t[1])]

def check_for_disconnected_fs_submissions(args: argparse.Namespace) -> None:
"""
Check for files without a corresponding Submission or Reply record in the
database.
"""
with app_context():
disconnected = find_disconnected_fs_submissions(args.store_dir)
if disconnected:
print(
"There are files in the submission area with no corresponding records in the
"
'database. Run "manage.py list-disconnected-fs-submissions" for details.'
)
else:
print("No unexpected files were found in the store.")
```

```
def list_disconnected_fs_submissions(args: argparse.Namespace) -> None:
"""
List files without a corresponding Submission or Reply record in the
database.
"""
with app_context():
disconnected_files = find_disconnected_fs_submissions(args.store_dir)
if disconnected_files:
print(
'Run "manage.py delete-disconnected-fs-submissions" to delete these files.',
file=sys.stderr,
)
for f in disconnected_files:
print(f)

def delete_disconnected_fs_submissions(args: argparse.Namespace) -> None:
"""
Delete files without a corresponding Submission record in the database.
"""
with app_context():
disconnected_files = find_disconnected_fs_submissions(args.store_dir)
bytes_deleted = 0
time_elapsed = 0.0
rate = 1.0
filecount = len(disconnected_files)
eta_msg = ""
for i, f in enumerate(disconnected_files, 1):
remove = args.force
if not args.force:
remove = input(f"Enter 'y' to delete {f}: ") == "y"
if remove:
filesize = os.stat(f).st_size
if i > 1:
eta = filesize / rate
eta_msg = f" (ETA to remove {filesize:d} bytes: {eta:.0f}s )"
print(f"Securely removing file {i}/{filecount} {f}{eta_msg}...")
start = time.time()
secure_delete(f)
```

```
file_elapsed = time.time() - start
bytes_deleted += filesize
time_elapsed += file_elapsed
rate = bytes_deleted / time_elapsed
print(
"elapsed: {:.2f}s rate: {:.1f} MB/s overall rate: {:.1f} MB/s".format(
file_elapsed, filesize / 1048576 / file_elapsed, rate / 1048576
)
)
else:
print(f"Not removing {f}.")

def were_there_submissions_today(
args: argparse.Namespace, context: Optional[AppContext] = None
) -> None:
with context or app_context():
something = (
db.session.query(Source)
.filter(Source.last_updated > datetime.datetime.utcnow() -
datetime.timedelta(hours=24))
.count()
> 0
)
count_file = os.path.join(args.data_root, "submissions_today.txt")
open(count_file, "w").write(something and "1" or "0")

def add_check_db_disconnect_parser(subps: _SubParsersAction) -> None:
check_db_disconnect_subp = subps.add_parser(
"check-disconnected-db-submissions",
help="Check for submissions that exist in the database but not the
filesystem.",
)
check_db_disconnect_subp.set_defaults(func=check_for_disconnected_db_submiss
ions)

def add_check_fs_disconnect_parser(subps: _SubParsersAction) -> None:
check_fs_disconnect_subp = subps.add_parser(
"check-disconnected-fs-submissions",
help="Check for submissions that exist in the filesystem but not in the
database.",
)
check_fs_disconnect_subp.set_defaults(func=check_for_disconnected_fs_submiss
ions)

def add_delete_db_disconnect_parser(subps: _SubParsersAction) -> None:
delete_db_disconnect_subp = subps.add_parser(
"delete-disconnected-db-submissions",
help="Delete submissions that exist in the database but not the
filesystem.",
)
delete_db_disconnect_subp.set_defaults(func=delete_disconnected_db_submissio
ns)
delete_db_disconnect_subp.add_argument(
"--force", action="store_true", help="Do not ask for confirmation."
)

def add_delete_fs_disconnect_parser(subps: _SubParsersAction) -> None:
delete_fs_disconnect_subp = subps.add_parser(
"delete-disconnected-fs-submissions",
help="Delete submissions that exist in the filesystem but not the
database.",
)
delete_fs_disconnect_subp.set_defaults(func=delete_disconnected_fs_submissio
ns)
delete_fs_disconnect_subp.add_argument(
"--force", action="store_true", help="Do not ask for confirmation."
)

def add_list_db_disconnect_parser(subps: _SubParsersAction) -> None:
list_db_disconnect_subp = subps.add_parser(
"list-disconnected-db-submissions",
help="List submissions that exist in the database but not the filesystem.",
)
list_db_disconnect_subp.set_defaults(func=list_disconnected_db_submissions)

def add_list_fs_disconnect_parser(subps: _SubParsersAction) -> None:
list_fs_disconnect_subp = subps.add_parser(
"list-disconnected-fs-submissions",
help="List submissions that exist in the filesystem but not the database.",
)
```

```
list_fs_disconnect_subp.set_defaults(func=list_disconnected_fs_submissions)

def add_were_there_submissions_today(subps: _SubParsersAction) -> None:
parser = subps.add_parser(
"were-there-submissions-today",
help=("Update the file indicating " "whether submissions were received in
the past 24h."),
)
parser.set_defaults(func=were_there_submissions_today)
```

Static Code Metrics:

Metric	Value
Lines of Code (LOC)	204
Comment Density	0.0
Cyclomatic Complexity	43
Maintainability Index	31.726125396795382
Halstead Metrics	
Unique Operators	9
Unique Operands	33
Total Operators	21
Total Operands	40
Volume	328.9313627895044
Effort	1794.171069760933
Depth of Inheritance Tree (DIT)	0
Coupling Between Object classes (CBO)	0
Lack of Cohesion of Methods (LCOM)	0
Fan-in	33
Fan-out	33
Number of Methods (NOM)	0

McCall's Intermediate Quality Metrics:

Metric	Value
Modifiability	0.15

Testability	0.23
Reliability	0.36
Understandability	0.29
Self-Descriptiveness	0.1
Reusability	1.0
Portability	0.6
Efficiency	0

GPT Analysis Report:

The provided Python code shows several areas that can be improved for better quality, maintainability, and readability. Below is a detailed analysis of specific areas alongside suggestions for improvement.

1. Comment Density and Documentation

- **Observation**: The comment density is reported as 0.0, which suggests that there are very few comments explaining the purpose and functionality of the code.
- **Suggestion**: Each function has a docstring, but they could include more detail on parameters and return types. It's recommended to use `docstring` conventions like [Google's Python Style Guide](https://google.github.io/styleguide/pyguide.html#383-docstrings) for consistency.

Example:

```
python
def find_disconnected_db_submissions(path: str) -> List[Submission]:
"""
Finds Submission records whose file does not exist.

Args:
path (str): The path where submission files are stored.

Returns:
List[Submission]: List of submissions missing from the file system.
"""
```

2. Cyclomatic Complexity

- **Observation**: With a cyclomatic complexity of 43, the code is relatively complex, indicating many conditional paths and loops, which can make it difficult to understand and maintain.
- **Suggestion**: Refactor complex methods by breaking them down into smaller, logically cohesive functions. For instance, the `delete_disconnected_fs_submissions` method could be separated into helper functions: one for prompting user confirmation, another for secure deletion, and one for logging progress.

3. Consolidation of Logic

- **Observation**: There are similar patterns in checking for disconnected submissions in both database and filesystem contexts. A lot of code duplication can be seen between these functionalities.
- **Suggestion**: Consider creating a generic function that abstracts the logic for checking disconnection whether from the DB or filesystem. This code reuse would reduce maintenance overhead.

```
python

def check_disconnected_records(source_list: List[str], file_map: Dict[str, bool]) -> List[str]:

return [f for f in source_list if f not in file_map]
```

4. Error Handling

- **Observation**: The code lacks error handling, especially around file operations or database queries which can fail due to various reasons (like permission issues or database connection failures).
- **Suggestion**: Implement try-except blocks wherever IO operations or database interactions are performed to handle exceptions gracefully and provide meaningful error messages.

```
python

try:

files_in_fs = {f: os.path.abspath(os.path.join(directory, f)) for f in files}

except Exception as e:

print(f"Error accessing filesystem: {e}")

return []
```

5. Input Handling

- **Observation**: Direct input collection using `input()` can lead to blocking behavior in the code.
- **Suggestion**: Consider using an argument parser for all options, or provide a way to refine user prompts. For instance, in a script meant for batch processing, passing flags would be more user-friendly and quicker.

6. Code Format and Consistency

- **Observation**: Code formatting is slightly inconsistent, especially around whitespace and line breaks.
- **Suggestion**: Utilize tools like `black` or `flake8` for Python code formatting to ensure consistency across the entire codebase.

7. Improve File and Submission Handling

- **Observation**: In the `find_disconnected_fs_submissions`, sizes are calculated but might not be necessary as they increase complexity and decrease performance when not needed.
- **Suggestion**: Simplify the discrepancies found in the filesystem—if the sole purpose is to identify and possibly delete files, size calculations can be excluded during the initial search process.

8. Function Names and Naming Conventions

- **Observation**: Function names are descriptive but can be simplified for clarity.
- **Suggestion**: While clarity is important, further simplification (without losing meaning) could help. For example, names like `check_for_disconnected_db_submissions` might be refactored to `check_db_disconnection`.

9. Testability Metrics

- **Observation**: A testability score of 0.228 suggests that the code is currently hard to test.
- **Suggestion**: Break functions down further, use dependency injection to facilitate testing, and structure functions to be more unit-test friendly. Mocking the database interactions will improve isolation during tests.

Conclusion

Refactoring the code based on the suggestions provided will greatly enhance its quality, maintainability, and readability, ultimately making it easier for future developers to work with the system. It’s also advisable to incorporate continuous integration practices where automated testing can be employed to check for any issues introduced during refactoring or feature development.