Birdry

Code Inspection Report

Team 1
Kyle Fritz
Laras Istiqomah
David Leiberg
Julian Sniffen
Nicholay Topin

Client
MAPLE Lab (point of contact: John Winder)

5/3/17

Birdry
Code Inspection Report

Table of Contents

	Page
1. Introduction	2
1.1 Purpose of This Document	2
1.2 References	2
1.3 Coding and Commenting Conventions	2
1.4 Defect Checklist	2
2. Code Inspection Process	3
2.1 Description	3
2.2 Impressions of the Process	4
2.3 Inspection Meetings	4
3. Modules Inspected	6
4. Defects	8
Appendix A – Team Review Sign-off	10
Appendix B – Document Contributions	11

1. **Introduction**

1.1 Purpose of This Document

The purpose of this document is to log the code inspection process for the Birdry application. The team has inspected the code for defects. The conventions and processes used, as well as the specific steps taken, are outlined here.

1.2 References

Throughout this document references will be made to:

1. The Birdry System Requirement Document
2. The Birdry System Design Document
3. Android's "Code Style for Contributors" (https://source.android.com/source/code-style)
4. Django's "Coding Style" (https://docs.djangoproject.com/en/dev/internals/contributing/writing-code/coding-style/)
5. Caffe's "Coding Style" (http://caffe.berkeleyvision.org/development.html)

1.3 Coding and Commenting Conventions

Because each of the three portions of the project served a different purpose and followed the best practices established by a different sub-field, separate coding and commenting conventions were adopted for the Android application, the server backend, and the neural network classifier. Where possible, we adopted existing coding standards for the technology being used. Specifically, we adhered to the official Android and Django coding and commenting conventions for the mobile application and Django server, respectively, as best we could (see references 3 and 4 in section 1.2). While these specifications were designed for developers contributing to Android and Django's code base, we decided that these standards, meant to regulate additions to much larger and more complex technologies, would provide adequate structure and format for our own, smaller application. Likewise, the neural network classifier was written following the standards of the framework used (see reference 5 in section 1.2). These consist of the Google Python and C++ coding standards and a separate standard for documenting network structure via a Prototxt file.

1.4 Defect Checklist

To guide our search for defects, we have compiled a list of potential errors and convention violations, listed in Table 1. Each of the defects belongs to one of the four primary categories we have identified: coding convention violations, logic errors, security oversights, and commenting conventions.

Table 1. Checklist of possible defects

	Category	**Defect**
1	Coding Convention	Variable not named based on relevant standard
2	Coding Convention	Method not named based on relevant standard
3	Coding Convention	Hard-coded value used instead of named constant
4	Coding Convention	Operations not performed in expected method as determined by relevant standard
5	Logic Error	Unreachable code
6	Logic Error	Wrong method called
7	Logic Error	Comparison performed using incorrect operator
8	Logic Error	Initialized value never used
9	Security Oversight	Exception improperly handled
10	Security Oversight	Bounds not checked during structure access
11	Commenting Convention	Old/unused code commented out without description of what code did and why it is commented out
12	Commenting Convention	Purpose of method not clear and no suitable comments included
13	Commenting Convention	"Todo" or "Fix me" comments still in place
14	Commenting Convention	Comments do not follow standards in regards to frequency and/or format
99	Other	Other defect of disposition not listed above

2. **Code Inspection Process**

2.1 Description

There were three different code bases for the team to inspect: the Android application code, the Django-database code, and the neural network code. The team focused more on inspecting the Android application code and the Django-database code, because we figured those were more susceptible or likely to have defects due to their relatively large size compared to the neural network code. The team used the informal "Over-the-Shoulder" review process, which consists of a preparation phase, an inspection meeting, a rework phase, and then a final completion phase, while reviewing the three code units.

While we were not able to have all team members present at once for any of the code inspections which took place, largely due to scheduling conflicts and general availability issues, we were able to always have four out of the five members conducting inspections.

2.2 Impressions of the Process

Overall, code inspection was beneficial to our team's progress and understanding of the application we have been developing. Because the various components of our application differ greatly in function, language, and other factors as described previously, coding for each component was done by separate subsets of team members. By performing code inspections, team members who had been working on one component were able to more clearly see the inner workings of all of the components of the application and how they were structured, and not just as black-box units with which to interface. Additionally, as some of the technologies used for our application, like Django and Android, were new to members of the team, code inspection allowed all members to be exposed to the stylings and composition of code written for all those technologies, and not just those on which they worked.

Despite the complexity of neural networks, we believe that the classifier portion of our backend has the least likelihood of containing any remaining flaws. Because existing work was followed for the architecture design and the corresponding Python driver is short, we believe there was little opportunity for error. Furthermore, the framework we used, Caffe, puts a lot of emphasis in automated testing of its own subcomponents, so a proper combination of components is not expected to fail. The fact that our classifier has functioned as intended is a sign that the subcomponents have been put together properly and a problem arising at this stage would be unlikely.

In contrast, the camera portion of the Android application is the most likely to have concealed flaws. We had to use a deprecated interface for technical reasons, and the use of this interface may lead to future problems. Though our inspection showed that the interface was used properly and the application was stable during testing, the application may not work as well with different cameras.

2.3 Inspection Meetings

To ensure all code adhered to relevant standards, a number of inspection meetings were held. The time, location, and purpose of each of our meetings are listed below.

Code Inspection Meeting #1: Initial meeting for Android application

Date: 27 April 2017
Time start: 5.15 pm
Time end: 5.40 pm

Location: UMBC Library Study Room 369
Participation:
- Kyle Fritz: inspector
- Laras Istiqomah: scribe
- David Leiberg: moderator
- Julian Sniffen: author

Code unit: Android application code

Code Inspection Meeting #2: Follow-up meeting for Android application
Date: 28 April 2017
Time start: 2.49 pm
Time end: 3.10 pm
Location: UMBC MAPLE Lab (ITE 339)
Members and roles:
- Laras Istiqomah: scribe
- David Leiberg: moderator
- Julian Sniffen: author
- Nicholay Topin: inspector

Code unit: Android application code

Code Inspection Meeting #3: Initial meeting for Django application
Date: 28 April 2017
Time start: 3.29 pm
Time end: 3.50 pm
Location: UMBC MAPLE Lab (ITE 339)
Members and roles:
- Laras Istiqomah: scribe
- David Leiberg: author
- Julian Sniffen: moderator
- Nicholay Topin: inspector

Code unit: Django code

Code Inspection Meeting #4: Follow-up meeting for Android application
Date: 29 April 2017
Time start: 4.00 pm
Time end: 4.28 pm
Location: UMBC MAPLE Lab (ITE 339)
Members and roles:
- Laras Istiqomah: scribe
- David Leiberg: moderator
- Julian Sniffen: author
- Nicholay Topin: inspector

Code unit: Android application code

Code Inspection Meeting #5: Django Code Inspection

Date: 29 April 2017
Time start: 4.40 pm
Time end: 4.58 pm
Location: UMBC MAPLE Lab (ITE 339)
Members and roles:

- Laras Istiqomah: scribe
- David Leiberg: author
- Julian Sniffen: inspector
- Nicholay Topin: moderator

Code unit: Django code

Code Inspection Meeting #6: Caffe Code Inspection
Date: 29 April 2017
Time start: 4.59 pm
Time end: 5.14 pm
Location: UMBC MAPLE Lab (ITE 339)
Members and roles:

- Laras Istiqomah: scribe
- David Leiberg: inspector
- Julian Sniffen: moderator
- Nicholay Topin: author

Code unit: Caffe code

3. **Modules Inspected**

All modules were completed in time for this pass of code reviews. Each module adheres to the structure presented for it in the Birdry System Design Document. Here we have listed the three primary components of Birdry, their composition in terms of classes, and the function of each class.

Android

- BlankFragment.java
  - A placeholder fragment that provides a transparent layer on which to place buttons
- CameraPreview.java
  - A class that allows the application to access the camera hardware on the android device
- GalleryFragment.java
  - The fragment that is responsible for displaying the images previously taken with the application
- GallerySortDialogFragment.java
  - A class that creates a dialog for the user to interact with
  - The dialog allows the user to sort the gallery over several predetermined options
- ImageAdapter.java
  - A helper class that aggregates the image data into one object for easy use

    - The class returns the images in a format that is consumed by the Gallery Fragment
- ImageItem.java
    - A class that encapsulates an image and its metadata
- MainActivity.Java
    - The driver for the application that runs when the application is first started
    - Responsible for displaying the camera to the user and getting user permissions
- MapFragment.java
    - The fragment that is responsible for displaying a map of previous images to the user
    - The fragment utilizes a Google API in order to display a map easily
- MapSortDialogFragment.java
    - A class that creates a dialog for the user to interact with
    - The dialog allows the user to sort the map markers over several predetermined options
- PictureDetailActivity.java
    - A class that allows a user to view, delete, and review an image

Django

- models.py:
    - Database models for storing information about pictures sent by users to the server
    - Allows Django to construct statements with the PostgreSQL database and store the information in the constructed models it contains
    - Contains additional field for "Picture" entity from original design, namely an additional Boolean indicating whether a picture is marked for review. Does not significantly alter original design considerations.
    - Implementation of "Entity-Relationship Model of Database", Figure 3 of System Design Document

- views.py:
    - Determines behavior of Django in handling communications (in the form of HTTP requests) between the backend server and the Android application
    - Part of "Django REST Framework" in Figure 1 of System Design Document

- settings.py:
    - Contains configuration options for Django server
    - Establishes use of PostgreSQL as database for information storage, use of REST Framework constructs for communication and authentication, debug options, etc.
    - Part of "Django REST Framework" in Figure 1 of System Design Document

- urls.py:
    - Contains configuration of available HTTP URL endpoints for the Django server to expose and receive communications on
    - Part of "Django REST Framework" in Figure 1 of System Design Document

Python/Caffe

- quickCode.py:
    - Driver for neural network classifier
    - Used to pass photographs from Django server to network and convert network output to a format usable by the Django server
    - Part of "ImageNet Neural Network" in Figure 1 of System Design Document
- net.prototxt:
    - Structure for neural network classifier
    - Defines the network used to classify photographs
    - Part of "ImageNet Neural Network" in Figure 1 of System Design Document

4. **Defects**

Violations in each code base are listed below, in Tables 2-4. Violations and their reference numbers are derived from those established in the violation checklist from section 1.4. The broad category of each violation has been included for convenience.

Table 2: Defects found in the Android application

Module	Violation [Code]	Description
PictureDetailActivity.java	Coding Convention [4]	Incorrect way to code OnClickListener, LocationListener. These are implemented inside of the code while the typical Android convention is to implement them on the same line as the class itself.
MainActivity.java	Commenting Convention [12]	Multiple functions without header comments explaining exactly what it is accomplishing.
MainActivity.java	Coding Convention [4]	locationManager.requestLocationUpdates() function is in the onCreate Android lifecycle function, when it should be in the onResume. Also, this function requires to check whether or not the location permission has been granted whenever it is accessed.

Table 3: Defects found in the Django server

Module	Violation	Description
models.py	Other	Written in Python 2.X instead of Python 3.

views.py settings.py urls.py	[99]	Needs to be changed in order to properly interface with the other Python (Caffe) code.
views.py	Logic Error [5]	Various return statements throughout will not fire due to being below earlier return statements.
views.py	Commenting Convention [13]	TODO comments present in code
views.py models.py urls.py	Coding Convention [1]	Variable names camel-cased (e.g. firstSecond) and not underscore-separated (first_second) as dictated by Django coding standards.
views.py	Coding Convention [4]	Import from disparate modules all on one line instead of on separate imports per Django coding standards
settings.py	Commenting Convention [11]	SQLite database engine constructs still present in comments, but not explained as to why commented out. Also goes for STATIC_URL and INSTALLED_APPS

Table 4: Defects found in the neural network system (Python/Caffe module)

Module	Violation	Description
quickCode.py	Commenting Convention [11]	Code used for unit testing is commented out, but its purpose is not described in comments. With the addition of these comments, the code is made useful to other testers.
quickCode.py	Coding Convention [3]	Image manipulations use fixed values for spatial dimensions, number of channels, etc. These need to be made named constants to make code easily readable and modifiable by others.

Appendix A – Team Review Sign-off

We, the members of the development team, undersign here to indicate that every member of the team has reviewed and approved the contents and format of this Code Inspection Report, and that their comments and/or concerns (if any) are listed here.

Name: ______________________ Signature: _______________________ Date: ________

Comments:

____

____

Name: ______________________ Signature: _______________________ Date: ________

Comments:

____

____

Name: ______________________ Signature: _______________________ Date: ________

Comments:

____

____

Name: ______________________ Signature: _______________________ Date: ________

Comments:

____

____

Name: ______________________ Signature: _______________________ Date: ________

Comments:

____

____

Appendix B – Document Contributions

Below is an estimate of the contribution from each team member, including specific work done and percentage of total document completed.

Kyle Fritz
<u>Contributions:</u>
- Document formatting and review
- Section 4 coauthor (Android portion)

<u>Est. Contribution:</u> 7%

Laras Istiqomah
<u>Contributions:</u>
- Section 2.1 author
- Section 2.3 author

<u>Est. Contribution:</u> 13%

David Leiberg
<u>Contributions:</u>
- Section 1.3 coauthor
- Section 2.2 coauthor
- Section 3 coauthor (Django portion)
- Section 4 coauthor (Django portion)

<u>Est. Contribution:</u> 30%

Julian Sniffen
<u>Contributions:</u>
- Section 3 coauthor (Android portion)
- Section 4 coauthor (Android portion)

<u>Est. Contribution:</u> 25%

Nicholay Topin
<u>Contributions:</u>
- Section 1.3 coauthor
- Section 1.4 author
- Section 2.2 coauthor
- Section 3 coauthor (Caffe portion)
- Section 4 coauthor (Caffe portion)

<u>Est. Contribution:</u> 25%