2021 FALL COEN6311 Assignment 2 Report

Jun Huang
40168167
Email: youyinnn@foxmail.com

I. T1: Changes Handling Process

"In addition to above description, history of the bike tracking can be queried by the user given a time span specified by the user. The history is kept maximum for a year. The user can choose the time interval between 1 minute, 5 minutes, to 1 hour to record the GPS locations of the bike."

As the description presented above, this change will not affect the existing system and functionalities. Hence it can be introduced into the system as an incremental feature with the idea of incremental development with no necessity of considering the refactoring of the existing implementation.

Fig. 1 presents the process of how to cope with such requirement change. The process can be described as the following steps:

1. **System re-modeling and re-design**: the models of the system should be re-design including:
   - Use case modeling of the interaction between users and the system;
   - System external context(architecture) diagram;
   - System internal architecture diagram;
   - System behavioral view of diagram;
2. **Implementation**: fist define the test case units with input and output patterns that match the demands of the changed requirement along with defined interface classes of abstract classes. And then implement the interfaces. And then test the implementation. Continuing this loop till all the defined test case units were passed.
3. **System testing**: perform all the system test case units.
4. **System Integration or Refactoring**: if the system test did not pass, then locate the broken point in the code, and commit integration or refactoring for eliminating the effects introduced by the changes.
5. **Relese the bata version to selected user**: deploy the new code to the production environment, but only push this update to certain users who are willing to take this update and provide feedback actively.
6. **Handle the feedback**: continues beta feedback support for maybe 2 weeks to a month, fix and patch any issues that related to this change.
7. **Relese the stable version to all user**: relese the new change to stable channel.

II. T2: Context View Modeling 1

"Our solution would implement a GPS tracking device accelerometer and gyroscopic sensors to detect relevant changes in the state of a parked bike and immediately notify users."

As already been presented at the submission of assignment 1, the external architecture of the system can be seen as a way of the context of the system. Then it can be described by context model as is shown in Fig 2.

A total of three context objects is involved in this product:

1. **Chip System**: an integrated chip that include all the hareware components. This chip is installed on the bike of users. The chip shall have the ability to communicate with a network endpoint.
2. **Back-end System**: the back-end system wich receive and response a network request send from chips or users' mobile application. Meanwhile, the server can also store the data send from chips.
3. **Mobile Application**: the companion application installed on users' mobile devices. This application can communicate with the back-end server and display all information to the user. Also, users can send instructions to the chips through this application.

III. T3: Context View Modeling 2

"In addition to above description, history of the bike tracking can be queried by the user given a time span specified by the user. The history is kept maximum for a year. The user can choose the time interval between 1 minute, 5 minutes, to 1 hour to record the GPS locations of the bike."

Fig. 3[1] present the context modeling of the above description. The explanation dataflow in the diagram are listed below:

1. The back-end server is quering data from the database system.
2. The database system is returning data to the back-end system.
3. The companion application transfer user's instructions to the back-end system.
4. The back-end system response the demanded data to the companion application.
6. The user select the time interval on the companion application.
5. The companion application display the result to the user.

As the product has a back-end server of its own design at the very first place, no new context or stakeholder will be introduced into the product. Since this new requirement can be handled with a set of new features introduced to the software of the back-end server and the mobile applications.

[1]Since the chip system is not directly involved in this description, it will not be presented in this diagram.

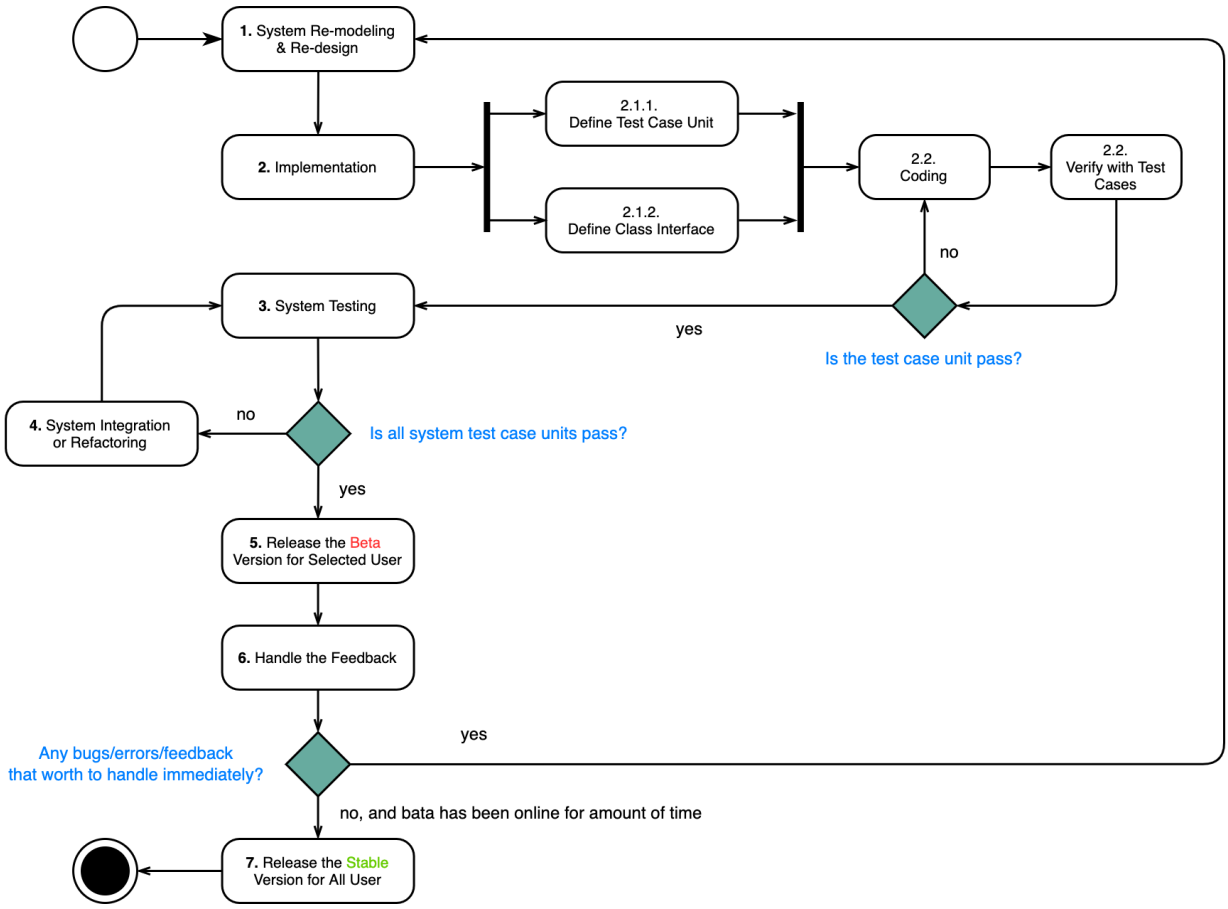

Fig. 1. Activity Diagram of Requirement Change Handling Process

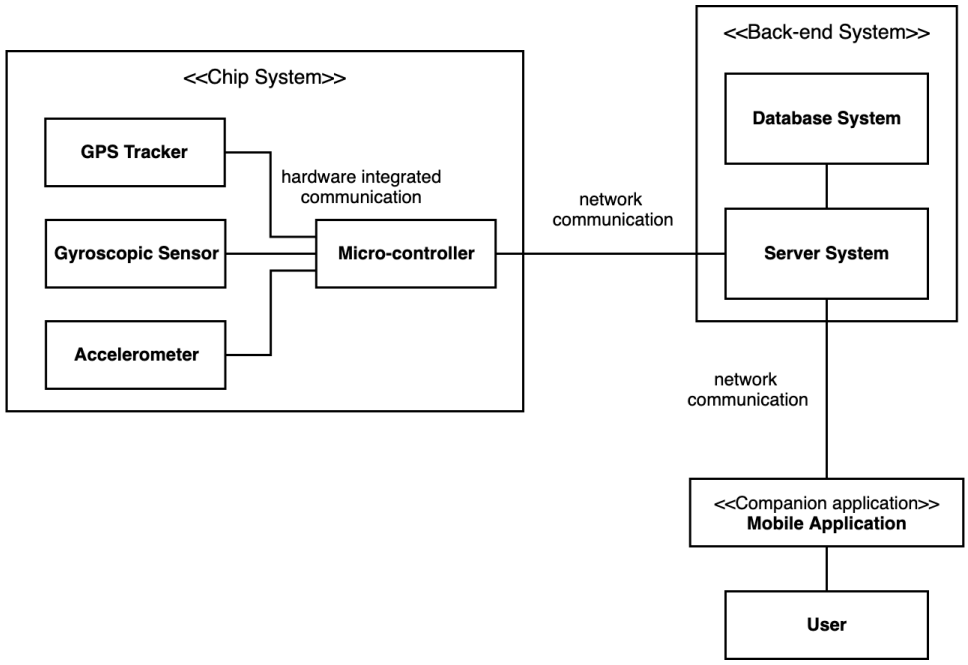

Fig. 2. Block Diagram of System Context of Communication

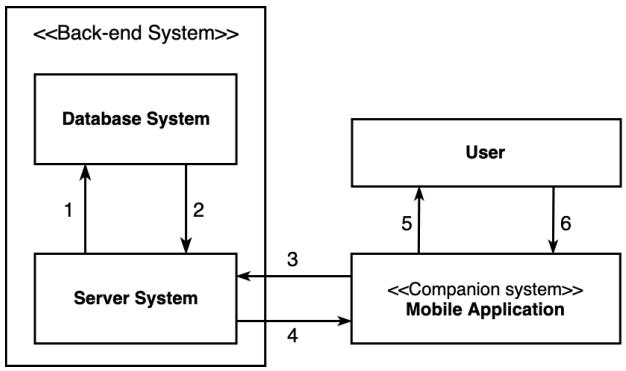

Fig. 3. Block Diagram of System Context of History Data Query

IV. T4: Behavioral View Modeling

"*The companion application will notify users of these changes through push notifications. The application will allow users to track their bike in the event immediate intervention is not possible.*"

Fig. 4 models the above description with an event-driven state diagram. The diagram represents the state of the chip system. Every chip will switch between those three states that are described below:

1. **"Parked" State**: the initial state when the user first installs the chip and initialized the chip and the companion application. Users can send an instruction to set this state back to the chip when the previous state is "Riding" state; Also the "AtRisk" state can be swith to this state when the gyroscopic data become stable. When the chips are at this state, the chips will automatically exam the gyroscopic data to estimate whether the chip is at the risk of stealing.
2. **"Riding" State**: the state that represents the bike was being used by the user. Users can inform the chip switch from the "Parked" state to this state. When the chips are at this state, the chips will not exam the gyroscopic data.
3. **"AtRisk" State**: the state that can be changed when recieving the unstable gyroscopic data are detected. When the chips are at this state, the chips will send notification to users' companion application.

V. T5: Observer-Subject Design Pattern Implementation

"*The companion application will notify users of these changes through push notifications. The application will allow users to track their bike in the event immediate intervention is not possible.*"

The modelling Class Diagram of applying the observer-subject design pattern to this scenario is shown in Fig. 5. This pattern is used at the back-end server context.

As Fig. 5 illustrates, the back-end server should manage a subject instance, and every user device should register on this subject as an observer. When the server receives the chip data from chips, it should notify the companion applications that installed on the users' devices.

The defined interfaces and the implementation are located at "/module/communication" package. The observer implementation is shown in Fig. 6. The subject implementation is shown in Fig. 7.

To complete this design pattern, a "MessageSender" interface that communicates with the users' devices and a "RiskDetector" interface that estimates the rick state are defined.

To perform the test cases unit for this design pattern, the interfaces described above are implemented in test folder as "MessageSenderTestImpl" and "RiskDetectorTestImpl" which are shown in Fig. 8 and Fig. 9.

The "MessageSenderTestImpl" use console printer as a representation of network communication with the users' devices. And the "RiskDetectorTestImpl" will estimate risk state by just force the state switch to "AtRisk" state manually.

The test case unit for all of these interfaces and implementation are presented at "io/github/youyinnn/module/communication/ChipDataObserverTest.java".

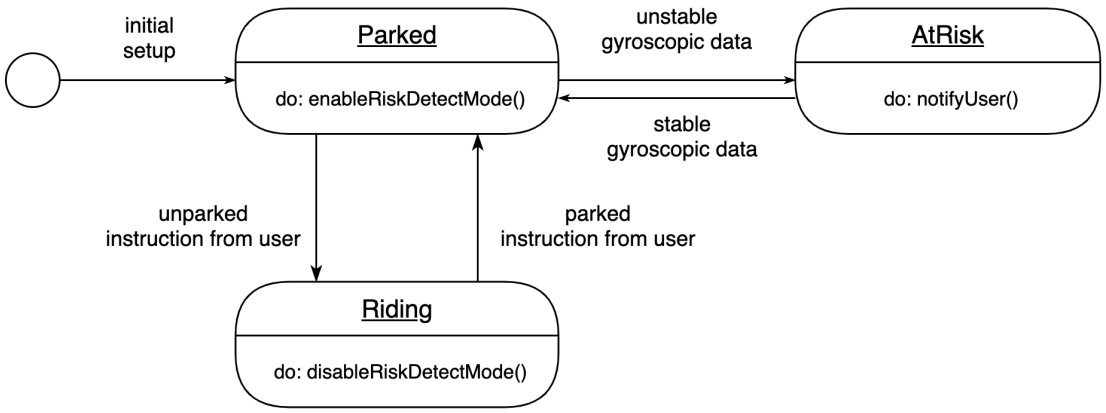

Fig. 4. State Diagram of the Chip System

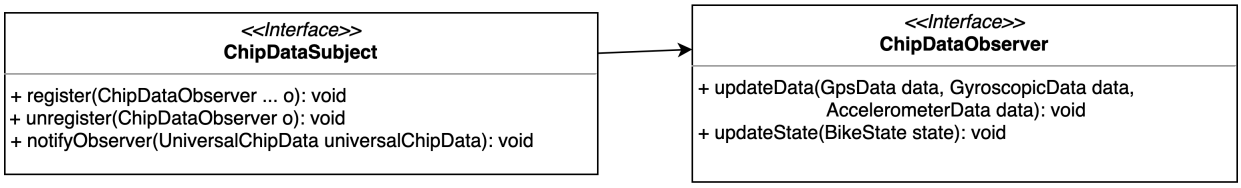

Fig. 5. Class Diagram of the Observer-Subject Design Pattern at the Back-end Server

```java
public class ChipDataObserverImpl implements ChipDataObserver {

    private final String chipNo;
    private final String deviceId;
    private final MessageSender messageSender;

    public String getChipNo() {
        return chipNo;
    }

    public ChipDataObserverImpl(String chipNo, String deviceId, MessageSender messageSender) {
        this.chipNo = chipNo;
        this.deviceId = deviceId;
        this.messageSender = messageSender;
    }

    @Override
    public void updateChipData(GpsData gpsData, GyroscopicData gyroscopicData,
                               AccelerometerData accelerometerData) {
        messageSender.sendData(deviceId, gpsData, gyroscopicData, accelerometerData);
    }

    @Override
    public void updateState(BikeState state) {
        if (state.equals(BikeState.AtRisk)) {
            messageSender.sendAtRiskAlert(deviceId);
        } else {
            messageSender.sendState(deviceId, state);
        }
    }
}
```

Fig. 6. ChipDataObserver Implementation

```java
public class ChipDataSubjectImpl implements ChipDataSubject {

    private static final HashMap<String, List<ChipDataObserver>> CHIP_AND_DEVICE_MAP
            = new HashMap<>(16);

    private final RiskDetector riskDetector;

    public ChipDataSubjectImpl(RiskDetector riskDetector) {
        this.riskDetector = riskDetector;
    }

    @Override
    public void register(ChipDataObserver ... obs) {
        for (ChipDataObserver o : obs) {
            ChipDataObserverImpl observer = (ChipDataObserverImpl) o;
            final String chipNo = observer.getChipNo();
            CHIP_AND_DEVICE_MAP.putIfAbsent(chipNo, new ArrayList<>(16));
            CHIP_AND_DEVICE_MAP.get(chipNo).add(o);
        }
    }

    @Override
    public void unregister(ChipDataObserver o) {
        ChipDataObserverImpl observer = (ChipDataObserverImpl) o;
        final String chipNo = observer.getChipNo();
        CHIP_AND_DEVICE_MAP.get(chipNo).remove(o);
    }

    @Override
    public void notifyObserver(UniversalChipData universalChipData) {
        final GpsData gpsData = universalChipData.getGpsData();
        final String chipNo = gpsData.getChipNo();
        final GyroscopicData gyroscopicData = universalChipData.getGyroscopicData();
        final AccelerometerData accelerometerData = universalChipData.getAccelerometerData();
        for (ChipDataObserver observer : CHIP_AND_DEVICE_MAP.get(chipNo)) {
            observer.updateChipData(gpsData, gyroscopicData, accelerometerData);
            if (riskDetector.isAtRisk(gyroscopicData)) {
                observer.updateState(BikeState.AtRisk);
            }
        }
    }
}
```

Fig. 7. ChipDataSubject Implementation

```java
public class MessageSenderTestImpl implements MessageSender {

    private static final Gson GSON = new Gson();

    @Override
    public void sendData(String deviceId,
                         GpsData gpsData,
                         GyroscopicData gyroscopicData,
                         AccelerometerData accelerometerData) {
        System.out.println(
            "\nSend" + "message to device: " + deviceId +
            "\n\tgps data:" + GSON.toJson(gpsData) +
            "\n\tgyroscopic data:" + GSON.toJson(gyroscopicData) +
            "\n\taccelerometer data:" + GSON.toJson(accelerometerData)
        );

        System.out.println(
            "\nDevice: " + deviceId + " received message: " +
                "\n\tgps data:" + GSON.toJson(gpsData) +
                "\n\tgyroscopic data:" + GSON.toJson(gyroscopicData) +
                "\n\taccelerometer data:" + GSON.toJson(accelerometerData)
        );
    }

    @Override
    public void sendState(String deviceId, BikeState state) {
        System.out.println(
            "\nSend state: " + state + "  to: " + deviceId
        );

        System.out.println(
            "\nDevice: " + deviceId + " received state: " + state
        );
    }

    @Override
    public void sendAtRiskAlert(String deviceId) {
        System.out.println(
            "\nSend risk alert to: " + deviceId
        );

        System.out.println(
            "\nDevice: " + deviceId + " received risk alert!!!"
        );
    }

}
```

Fig. 8. MessageSender Testing Implementation

```
public class RiskDetectorTestImpl implements RiskDetector {

    private static final HashMap<String, Boolean> CHIP_IS_AT_RISK = new HashMap<>(16);

    public void assertAtRisk(GyroscopicData gyroscopicData) {
        CHIP_IS_AT_RISK.put(gyroscopicData.getChipNo(), Boolean.TRUE);
    }

    @Override
    public boolean isAtRisk(GyroscopicData gyroscopicData) {
        return CHIP_IS_AT_RISK.getOrDefault(gyroscopicData.getChipNo(), Boolean.FALSE);
    }

}
```

Fig. 9. RiskDetector Testing Implementation