Evaluation of Test Plan and Instrumentation

Risks to be considered:

- Personnel risk: In the XP life cycle I plan to design unit and system tests straight away before I even begin code, as I am inexperienced in software testing this may mean that this initial stage takes far longer than usual leading to delays in code being developed and potential scheduling issues.
    - In order to mitigate this, I have included training time in the plan and aim to have a high level of competence before I start programming.
- Scheduling risk: as discussed above I will have to have a lot of training time in order to be able to test effectively. Despite planning for this, there will be many times where I encounter an issue and need more training time.
    - In order to mitigate this, I will include an hour each week in the plan, which is left as blank space, this space can be used for training or if none is needed, simply to cover other planned activities overspill.
- Test execution risk: as I am inexperienced in software testing there is a strong possibility that test execution for some areas will take far longer than expected, this would put significant strain on my resources and affect my schedule.
    - In order to mitigate this, I have kept more complicated system tests to a minimum and will focus on through unit and integration testing throughout development.
- Technology risk: I am also inexperienced with java/object-oriented programming, web servers and Apache database. This may lead to a high error rate due to unfamiliarity with the tools I am using. Especially initially as I learn my way through it.
    - In order to mitigate this, I will schedule a further hour each week to be used for training on the software I am using and for testing unfamiliar systems.
- Scheduling Risk: As we are testing runtime late in development there is the risk that we will uncover issues that are causing slow runtime late and these will be expensive to go back and fix as we may have to retest whole areas of code and retest integration too.
    - In order to mitigate this, we should implement some runtime testing throughout, to ensure that there aren't any glaring obvious issues.
- Development Risk: As we will be using synthetic data to test the runtime of the system in the "system testing" phase, there is the risk that this data is unrepresentative and could give us a difference in expected behavior and actual behavior in the real world which would lead to poor quality in the "release phase".
    - This is fairly likely as we may misrepresent which shops are popular, what types of items people order etc. as these can be hard to predict.
    - However, these changes in the data shouldn't significantly change the runtime if it is sufficiently low.
    - In order to mitigate this, we should validate the data by having real people make lunch orders and using this to alter/create the synthetic data.

Evaluation of Test Plan

Chosen lifecycle:

The choice of Extreme Programming overall is well suited to the project; however, I did have to make some alterations in order to be able to create a testing plan. There are two notable omissions; Acceptance testing and incremented release. Unfortunately, due to the nature of my project I cannot effectively do acceptance testing as it must be a solo project (I can't show other people) and I cannot get feedback from the Informatics department (as this would affect grading) I have instead opted for full system testing using synthetic data. This should still give a good idea of the quality of the software but does represent a major limitation of the plan as the system cannot be tested with real users. This synthetic data may not be fully representative and therefore may be misleading as to the actual performance of the software.

Similarly, the omission of incremental release presents a limitation of the plan. Releasing (submitting) the software all in one go means that testing up to this point must be more thorough as once released I won't have the opportunity to fix any errors that arise.

Test plan requirements:

This section (while not exhaustive for all requirements) shows a strong plan for specific requirements and how to fit them into the XP lifecycle. It shows that often more complicated requirements will have to be tested at various points in the lifecycle in order to ensure reliability. It discusses the level of importance of each requirement and subsequently the amount of resource that should be allocated to them.

Then the test plan specifies Input, Output, Specification, this section sets out the variables that will be needed in order to create the tests, one criticism of the test plan is that it could go into more detail in this section in order to aid the testing process, it could go further, however I think that the level of detail is sufficient.

Evaluation of Instrumentation

Before Testing

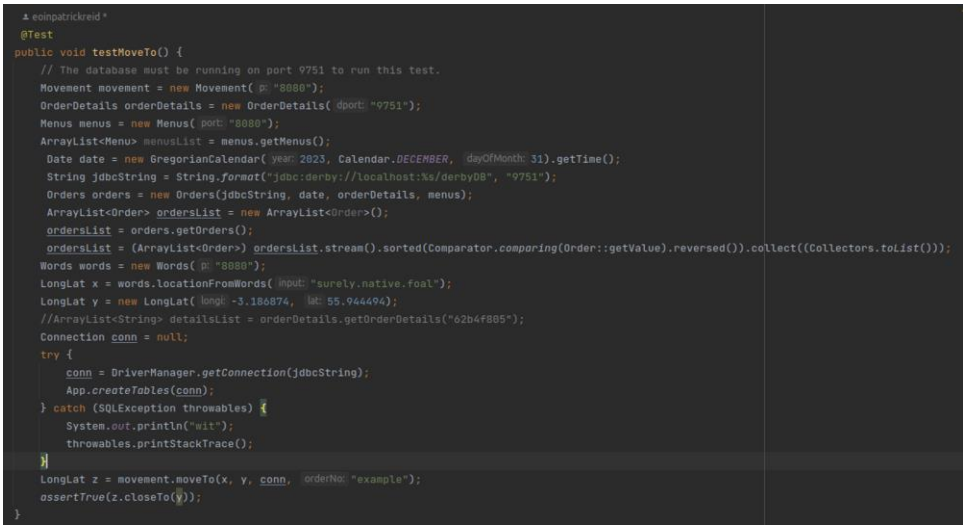
The test planning document gives an account of the scaffolding that is likely to be needed and the form it will take. It tells us the stage it is likely to be needed. However, one limitation of this section is that it is heavily dependent on the timings of development of certain sections of code – meaning that the level of instrumentation that is needed in practice may vary from this plan quite a lot.

Say for example for R2, it may be the case that the majority of the functionality required has been implemented and tested, then only a limited amount of instrumentation would be needed, and this requirement could be quickly tested. If, however, development has been delayed and other functionality hasn't been properly implemented/tested then this testing may take a long time causing scheduling issues.

After Testing

The above was written before I had implemented the tests, having now implemented the tests I think I underestimated the level of scaffolding needed for many of them. I didn't have a great understanding of how scaffolding worked and didn't fully appreciate the connectiveness of all the different functionalities.

This meant that when running the tests far more instrumentation was needed. The planning was correct in the need for simulated data in many of the functions. But it didn't go into enough detail in terms of the extensive instrumentation needed for some of the tests. For example



```
eoinpatrickreid *
@Test
public void testMoveTo() {
    // The database must be running on port 9751 to run this test.
    Movement movement = new Movement( p: "8080");
    OrderDetails orderDetails = new OrderDetails( dport: "9751");
    Menus menus = new Menus( port: "8080");
    ArrayList<Menu> menusList = menus.getMenus();
     Date date = new GregorianCalendar( year: 2023, Calendar.DECEMBER,  dayOfMonth: 31).getTime();
     String jdbcString = String.format("jdbc:derby://localhost:%s/derbyDB", "9751");
     Orders orders = new Orders(jdbcString, date, orderDetails, menus);
     ArrayList<Order> ordersList = new ArrayList<Order>();
     ordersList = orders.getOrders();
     ordersList = (ArrayList<Order>) ordersList.stream().sorted(Comparator.comparing(Order::getValue).reversed()).collect((Collectors.toList()));
    Words words = new Words( p: "8080");
    LongLat x = words.locationFromWords( input: "surely.native.foal");
    LongLat y = new LongLat( longi: -3.186874,  lat: 55.944494);
    //ArrayList<String> detailsList = orderDetails.getOrderDetails("62b4f805");
    Connection conn = null;
    try {
        conn = DriverManager.getConnection(jdbcString);
        App.createTables(conn);
    } catch (SQLException throwables) {
        System.out.println("wit");
        throwables.printStackTrace();
    }
    LongLat z = movement.moveTo(x, y, conn,  orderNo: "example");
    assertTrue(z.closeTo(y));
}
```

This was the instrumentation needed for moveTo to be tested adequately. In hindsight had I known how much instrumentation would be needed then I would have changed how I planned my time, scheduling more time on creating the scaffolding for tests.