THE UNIVERSITY OF SYDNEY

School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: SOFT2412

Assignment name: Tools for Agile Software Development

Tutorial time: Thursday 18:00-20:00 **Tutor name:** Ge Jin

DECLARATION
We the undersigned declare that we have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure*, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that understand that failure to comply with the *Student Plagiarism: Coursework Policy and Procedure* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Project team members				
Student name	**Student ID**	**Participated**	**Agree to share**	**Signature**
1. Xiaoqian Hu	500105263	Yes / No	Yes/No	Xiaoqian Hu
2. Haolin Jin	510085113	Yes / No	Yes / No	Jin JJL
3. Zechao Sun	510048657	Yes / No	Yes / No	Zechao Sun
4. Yiheng Yang	510231499	Yes / No	Yes / No	Yiheng Yang
5.		Yes / No	Yes / No	
6.		Yes / No	Yes / No	
7.		Yes / No	Yes / No	
8.		Yes / No	Yes / No	
9.		Yes / No	Yes / No	
10.		Yes / No	Yes / No	

Level 2, SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.info@sydney.edu.au
sydney.edu.au/it

ABN 15 211 513 464
CRICOS 00026A

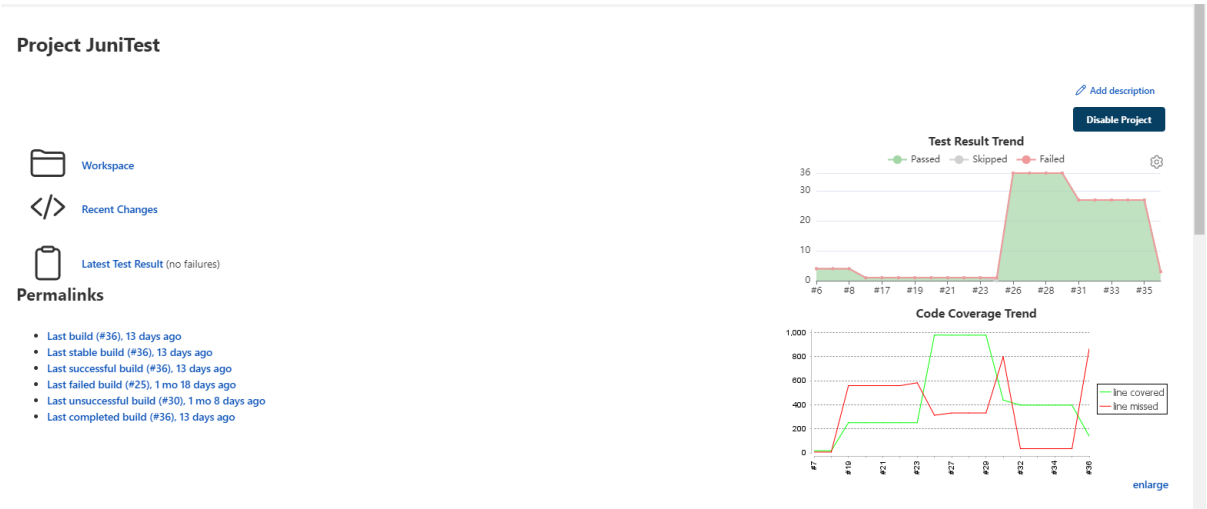
SOFT2412 Assignment2 Sprint2

Name	SID
Yiheng Yang	510231499
Zechao Sun	510048657
Haolin Jin	510085113
Xiaoqian Hu	500105263

1. Scrum group collaboration

In the second sprint, we mainly focused on the transaction function of our vending machine, and also set up the agile tools Jenkins to achieve the auto testing and building. Our scrum roles are exactly the same as the previous sprint, Yiheng Yang is the Product owner, Haolin Jin is the Scrum master and Xiaoqian Hu and Zechao Sun are core team members.

Our first meeting started on 22 October, we discussed the user stories and main goals for the second sprint. Then we set up the CI/CD pipelines for our project and make sure it achieves the auto building and testing, then we start working on the transaction system and keep adding subtasks for this user story. After we finish the user stories of the transaction function, we also decide to complete the functionality of the cashier, so add this into our product backlog as well in the second sprint.



Our second meeting started on 25 October, before we attended the second meeting we expected to complete the transaction functionality and cashier's functionality but in reality the cashier left one report generation function and the transaction should keep each transaction into the history file when it finished. Because there are only two days left for the second sprint, we worked together during the scrum meeting and spent some time on it. During the second sprint Zechao Sun is

responsible for the testing of our user stories, so after we finish all the user stories in the second sprint, we double check each test case and make sure it passed.

Our third meeting is on 27 October, we didn't do any coding during the meeting, we generally prepared for the demonstration and individual quiz part.

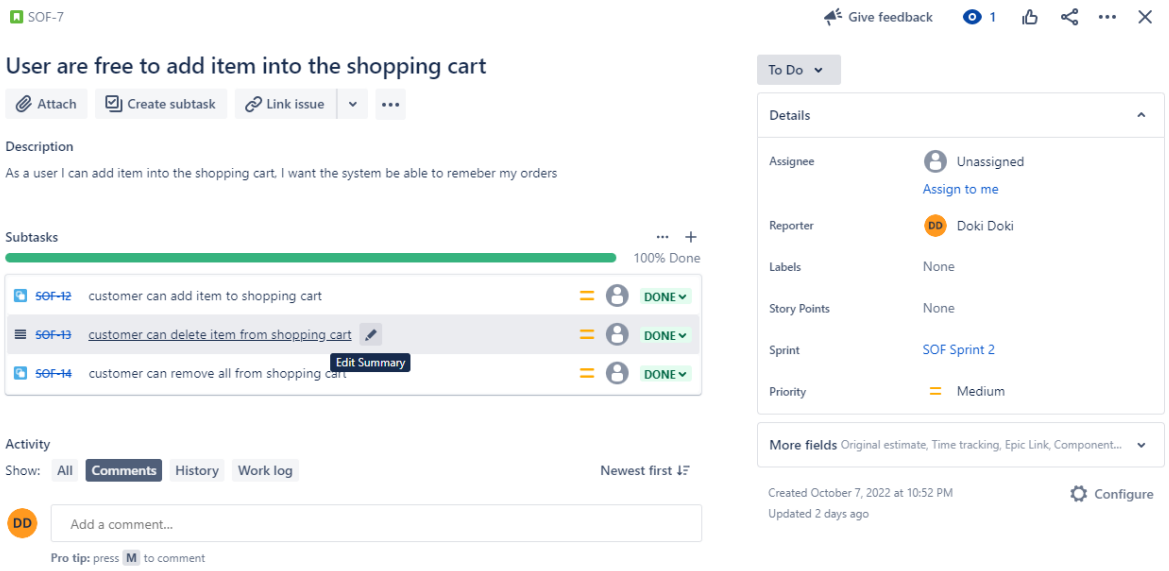
GithubLink: SOT2412-COMP9412-2022S2/T22_G03_Assignment2 (sydney.edu.au)

Jira Link:SOF board - Agile Board - Jira (atlassian.net)
(I will put the screenshot of the project board down below)

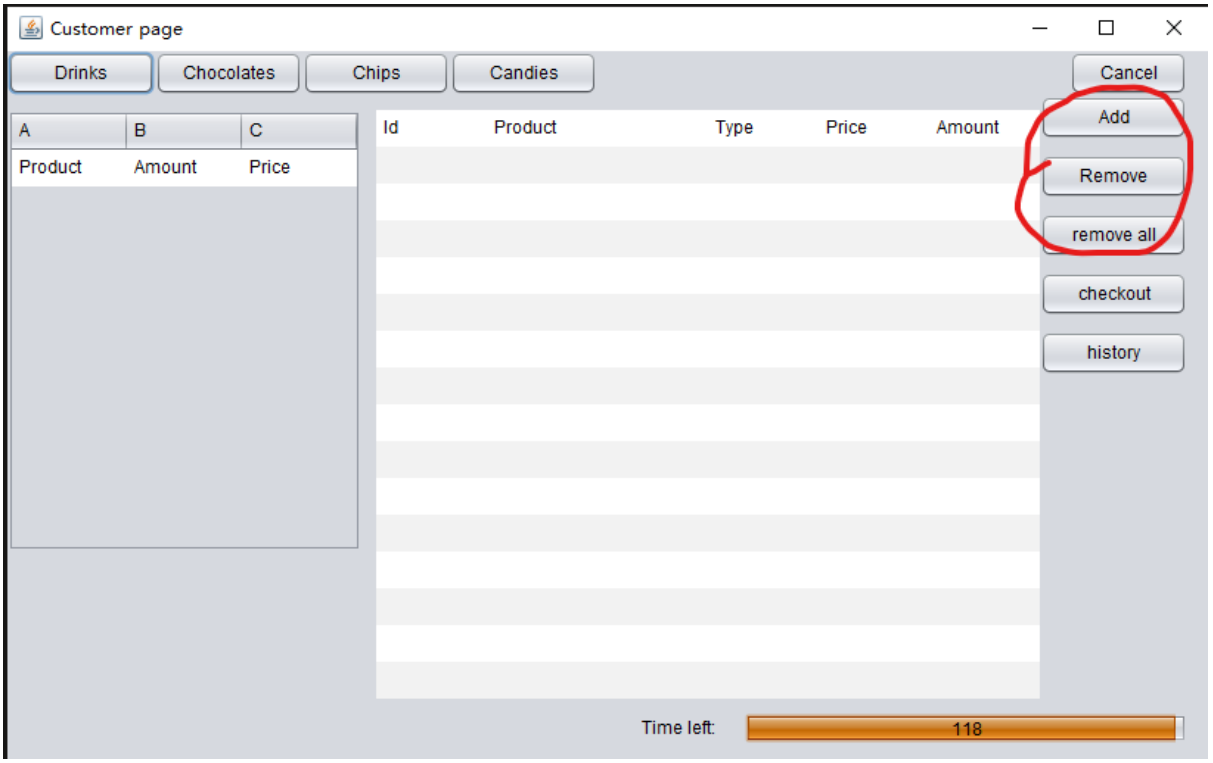
2. Planning and Goals

Because our first sprint is mainly focused on the scrum team setting up and preparing the product backlog, also the basic interface without the actual algorithms inside, so we decided to focus mainly on the transaction algorithm and shopping cart functionality this time.

The above planning is too general, because for the transaction functionality it includes shopping cart checkout and payment method ect. So it might be a challenge to achieve especially to implement the user interface using java swing. So we need to carefully split the goal into different tasks just like the user story, and assign them into different group members.

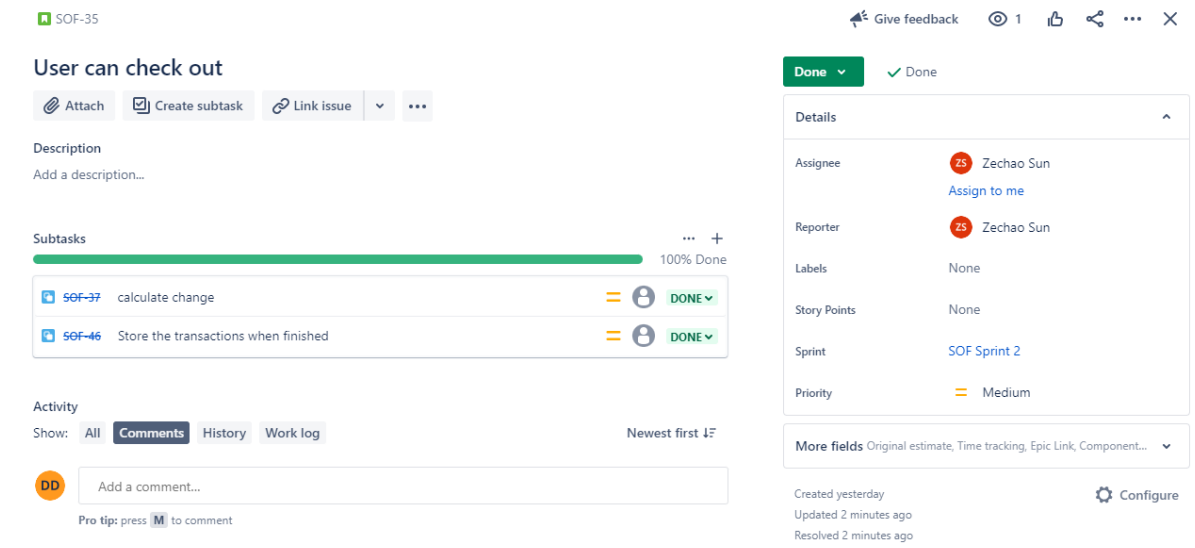


Just like the above screenshot, our first task is to set up the shopping cart for the transaction system, and we have three subtasks for it. We assign this task to Xiaoqian Hu who is also a core team member because he wrote some parts of the purchase system before. The expected outcome should be the user is able to interact with the shopping cart class properly via several buttons just like the screenshot below.



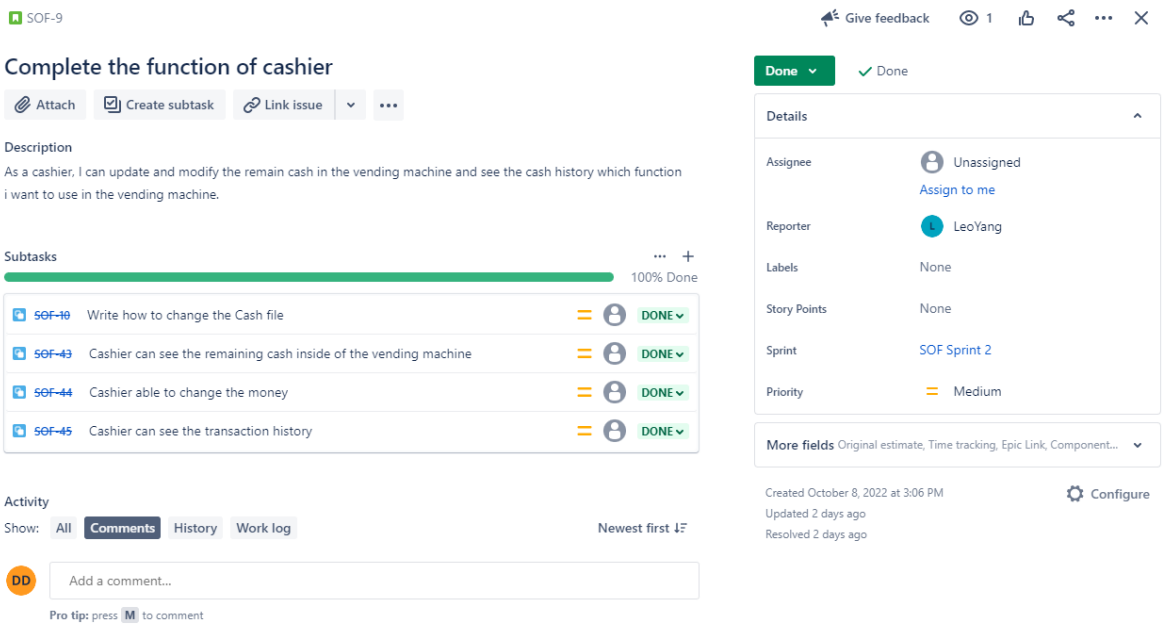
So when the user clicks “Add” it should add the product into the shopping cart and remove the selected product inside of the shopping cart.

Then we assign the change process after the transaction to “Haolin Jin” and “Yiheng Yang”



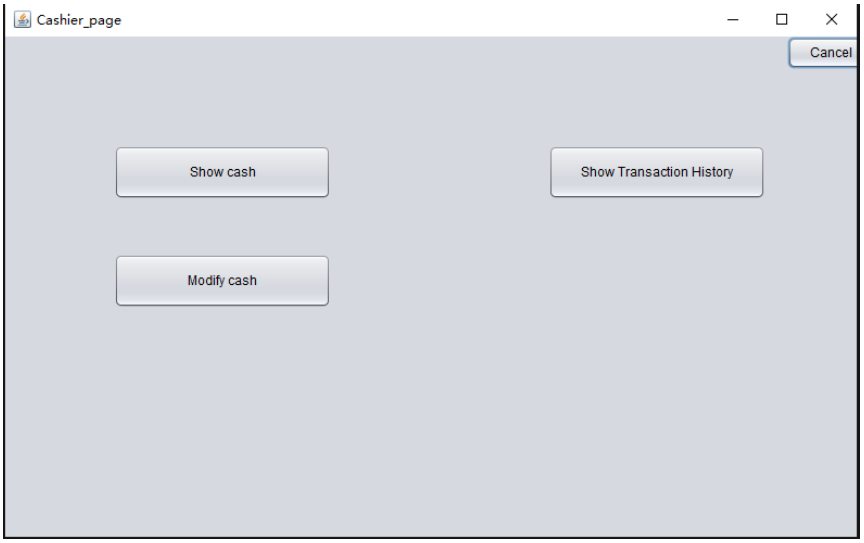
Just like the above image, there are two subtasks: the change algorithm and save functionality after each transaction. So at the end of sprint two, the user should get change and save their purchase history into the file.

The transaction is correlated to the cashier because the cashier is able to view the purchase history, so our second goal is to finish the cashier's function but jump over the login part because we decide to implement the login function when we dealing with the product owner.



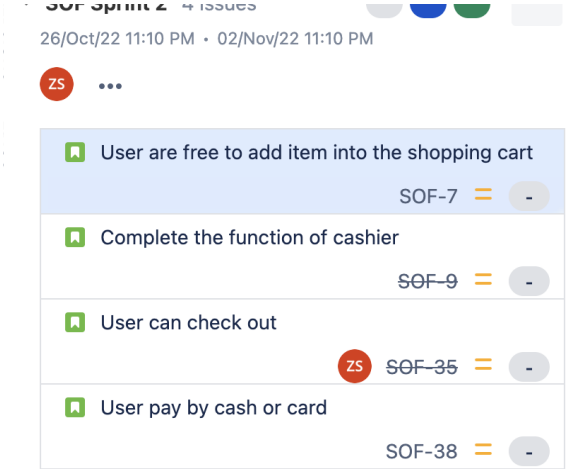
In this part, Haolin Jin was responsible for the transaction history part, and Yiheng Yang responsible for the rest of them. The most challenging part for this user story is how to generate two reports and read the transaction history from a file and put it onto the user interface. Also for every transaction the cashier's report should also be changed.

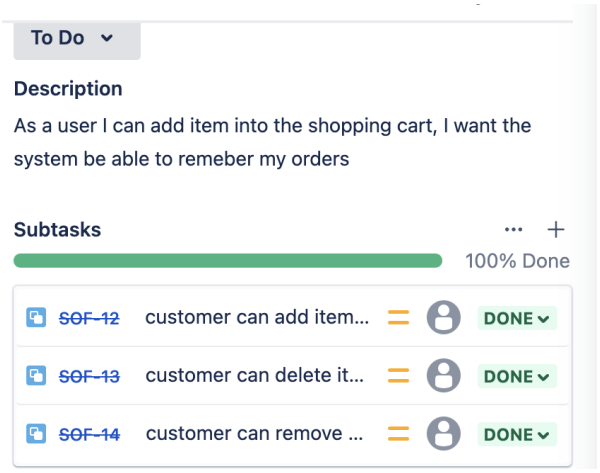
UI diagram for Cashier:



3. User story test cases

User story one:





Test cases for user story:

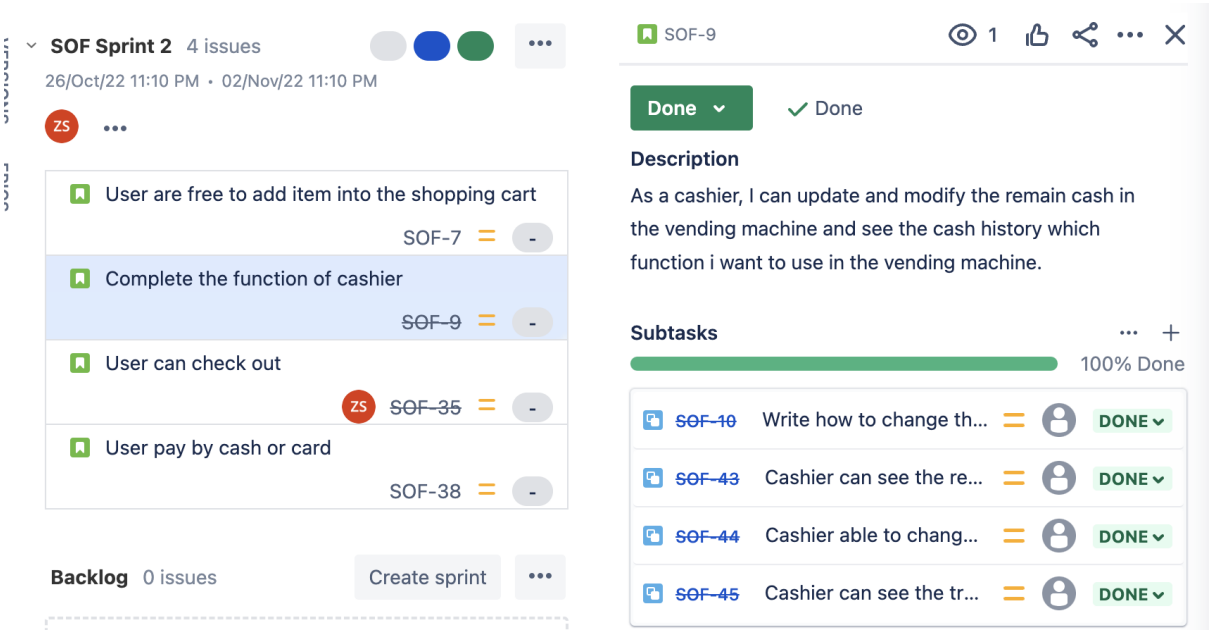
```java
@Test
public void TestAddToShoppingCart() {

    Customer c = new Customer( balance: 100,  id: 1,  username: "zechao",  password: "sunzechao2000");

    Snacks sn = new Snacks( name: "cola",  price: 5,  amount: 10,  id: 1,  type: "drinks");
    // test normal
    int real = c.addToShoppingCart(sn,  amount: 5);
    assertEquals( expected: 0, real);
    Snacks real_key = null;
    int real_value = 0;
    for (Map.Entry<Snacks, Integer> m : c.getShoppingCart().entrySet()) {
        real_key = m.getKey();
        real_value = m.getValue();
    }
    // case shoppingCart do not have this snack
    assertEquals(sn, real_key); // test object
    assertEquals( expected: 5, real_value); // test amount

    // case shoppingCart already have this snack and update amount

    c.addToShoppingCart(sn,  amount: 3);
    for (Map.Entry<Snacks, Integer> m : c.getShoppingCart().entrySet()) {
        real_key = m.getKey();
        real_value = m.getValue();
    }

    assertEquals(sn, real_key); // test object
    assertEquals( expected: 8, real_value); // test amount

    //edge case
```

This test case tests whether customers can add items to the shopping cart normally. An edge test case which tests if there is no snack's category for input snack. And another edge test case which is a shopping cart already has this snack and update amount. We passed this test case and the shopping cart can run normally.

```java
孙泽超 +1
@Test
public void testRemoveFromSC() {
    Customer c = new Customer( balance: 100,  id: 1,  username: "zechao",  password: "sunzechao2000");
    Snacks sn = new Snacks( name: "cola",  price: 5,  amount: 10,  id: 1,  type: "drinks");
    c.addToShoppingCart(sn,  amount: 9);
    //test snack does not exist in shoppingCart
    Snacks test_sn = new Snacks( name: "lemon juice",  price: 5,  amount: 10,  id: 1,  type: "drinks");
    int real = c.removeFromShoppingCart(test_sn,  amount: 5);
    assertEquals( expected: -2, real);

    //test input amount <= 0
    int real1 = c.removeFromShoppingCart(sn,  amount: 0);
    int real2 = c.removeFromShoppingCart(sn,  amount: -1);
    assertEquals(real1,  actual: -1);
    assertEquals(real2,  actual: -1);


    // test input amount > snack's amount
    int real3 = c.removeFromShoppingCart(sn,  amount: 10);
    assertEquals( expected: -3, real3);

    //test remove someS

    c.removeFromShoppingCart(sn,  amount: 3);
    Snacks real_key = null;
    int real_value = 0;
    for (Map.Entry<Snacks, Integer> m : c.getShoppingCart().entrySet()) {
        real_key = m.getKey();
        real_value = m.getValue();
    }
    assertEquals(sn, real_key);
    assertEquals( expected: 6, real_value);
```

This test case tests whether customers delete items from the shopping cart normally. I set a normal test case, and an edge test case which removes an item from a shopping cart that does not have this item. Another test case is about the input amount bigger than the snack's amount, and remove one snack and remove some snacks.We pass this test and the shopping cart can run normally.

User story two:



Test cases for user story:

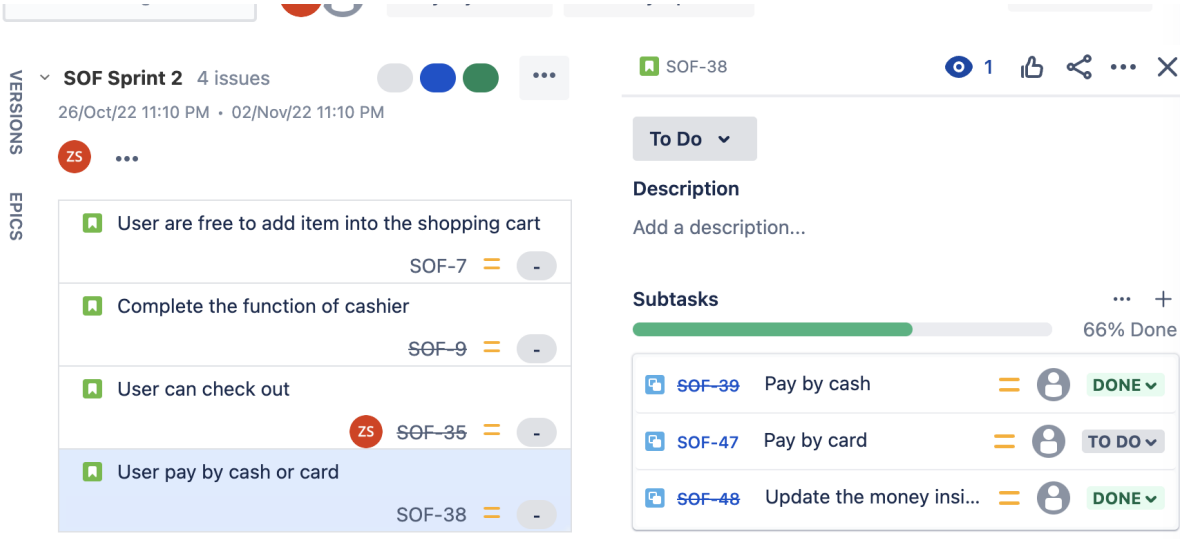
```
孙泽超
@Test
public void TestClearShoppingCart() {

    Customer c = new Customer( balance: 100,  id: 1,  username: "zechao",  password: "sunzechao2000");
    Snacks sn = new Snacks( name: "cola",  price: 5,  amount: 10,  id: 1,  type: "drinks");
    Snacks sn1 = new Snacks( name: "fake cola",  price: 5,  amount: 10,  id: 1,  type: "drinks");
    c.addToShoppingCart(sn,  amount: 9);
    c.addToShoppingCart(sn1,  amount: 9);
    c.clearShoppingCart();
    assertFalse(c.getShoppingCart().containsKey(sn));
    assertFalse(c.getShoppingCart().containsKey(sn1));
}
```

This test case tests if customers can clear shoppingCart normally. We passed this test case and our clear shopping cart function can run normally.

```java
@Test
public void TestGiveChange(){
    Cashier c = new Cashier( username: "szc", password: "szc2002");
    Map<Double,Integer> realMap = new HashMap<>();
    realMap = c.the_change_amount( amount: 10, total_price: 3.6);
    Map<Double,Integer> ExpectMap = new HashMap<>();
    ExpectMap.put(2.0,0);
    ExpectMap.put(1.0,1);
    ExpectMap.put(0.5,0);
    ExpectMap.put(0.2,2);
    ExpectMap.put(0.1,0);
    ExpectMap.put(0.05,0);
    ExpectMap.put(20.0,0);
    ExpectMap.put(10.0,0);
    ExpectMap.put(5.0,1);
    ExpectMap.put(100.0,0);
    ExpectMap.put(50.0,0);
    assertEquals(ExpectMap,realMap);

}
```

This test case tests if the system can change money correctly. we pass this test case and can change money correctly.For example, the output about the number of cash for each value is correct.

User story three:



Test cases for user story:

```java
@Test
// update the money inside of vending machine
public void TestUpdateMoney(){
    Cashier c = new Cashier( username: "szc", password: "szc2002");
    Map<Double,Integer> origin = new HashMap<>();
    origin.put(100.0,20);
    origin.put(50.0,30);
    origin.put(20.0,10);
    origin.put(10.0,50);
    origin.put(5.0,60);
    origin.put(2.0, 40);
    origin.put(1.0,35);
    origin.put(0.5,58);
    origin.put(0.2,45);
    origin.put(0.1,23);
    origin.put(0.05,49);
    //c.update("src/main/resources/TestCash");
    Map<Double,Integer> TestInput = new HashMap<>();
    TestInput.put(100.0,0);
    TestInput.put(50.0,0);
    TestInput.put(20.0,0);
    TestInput.put(10.0,0);
    TestInput.put(5.0,1);
    TestInput.put(2.0,0);
    TestInput.put(1.0,1);
    TestInput.put(0.5,0);
    TestInput.put(0.2,2);
    TestInput.put(0.1,0);
    TestInput.put(0.05,0);
```

This test case is about a cash file updated. If the user finishes the transaction, the system will provide the optimized changed money depending on the higher value principle. From the result the money's amount inside of the vending machine will change successfully which matched the above user story, so we can say this user story is achieved successfully.

```
    Map<Double,Integer> real = c.updateCash(TestInput, path: "src/main/resources/TestCash");
    Map<Double,Integer> ExpectMap = new HashMap<>();
    ExpectMap.put(100.0,20);
    ExpectMap.put(50.0,30);
    ExpectMap.put(20.0,10);
    ExpectMap.put(10.0,50);
    ExpectMap.put(5.0,59);
    ExpectMap.put(2.0,40);
    ExpectMap.put(1.0,34);
    ExpectMap.put(0.5,58);
    ExpectMap.put(0.2, 43);
    ExpectMap.put(0.1,23);
    ExpectMap.put(0.05,49);
    c.setCash(origin);
    c.update( path: "src/main/resources/TestCash");
    assertEquals(ExpectMap,real);
```

```
    //Test how to change the Cash file
    孙泽超
    @Test
    public void TestModifyCash(){
        Map<Double,Integer> origin = new HashMap<>();
        origin.put(100.0,20);
        origin.put(50.0,30);
        origin.put(20.0,10);
        origin.put(10.0,50);
        origin.put(5.0,60);
        origin.put(2.0, 40);
        origin.put(1.0,35);
        origin.put(0.5,58);
        origin.put(0.2,45);
        origin.put(0.1,23);
        origin.put(0.05,49);
        Cashier c = new Cashier( username: "szc", password: "szc2002");
        c.setCash(origin);
        c.modifyCash( unit: 2.0, quantity: 2,  path: "src/main/resources/TestModifyCash");
        Map<Double,Integer> real = c.getCash( path: "src/main/resources/TestModifyCash");
        int realAmount = real.get(2.0);
        assertEquals( expected: 2,realAmount);

        c.setCash(origin);
        c.update( path: "src/main/resources/TestModifyCash");

    }
}
```

This test case tests whether the cashier can modify or update Money normally, and if it can update the cash file. We passed this test case

because we can modify the cash file correctly and can show the right cash information on the screen.

4.Scrum retrospective

From the last sprint, our goals were too high so we removed some user stories and focused on the basic user interface. So at this time we try to avoid this issue and just set up two goals for the second sprint, and optimize our Jira board a bit.

For the scrum event part, I think the main reason of aboved issue is the lack of scrum review and planning, so in the second sprint we totally have 3 different meetings and a lot of consultation about the transaction functionality and code update using the github, so the efficiency will increase a lot during the second sprint, and each of team members have their own work to do.

6. Daily Scrum

Everyday we will have a 15-minute time-boxed event for the Development Team to synchronize activities and check whether we need to merge any conflict and plan for the next day. We usually did this through the zoom meeting which created by Haolin Jin who is the scrum master of this project and we found it quite useful, because we are responsible for the different part and it’s important to have around 15 minutes chat to reconnect our idea together and to make sure no one get out of the track.

5. Client feedback and review

We received several feedback during the second sprint demonstration.

- Implement login session for each role

In our system, there are no login sessions for cashiers and sellers, and we will work on this part in the third sprint.

- Optimize the change hashmap

In our current change system, the UI will display the whole hashmap of the change map which is not easy to understand for a normal client, so we should optimize this part maybe during the third sprint.