Task 1

Software development has always been hard. In a famous essay entitled "No Silver Bullet – Essence and Accidents in Software Engineering", more than 30 years ago Fred Brooks wrote:

> *"There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity"*

There is ample agreement that the situation is largely unchanged if one uses traditional forms of software development.

Task 1(a)

Explain what the reasons/causes are for this situation

There are a number of reasons and causes as to why the situation remains largely unchanged when using traditional forms of software development. This is mainly due to the actual nature of software engineering, which in essence is extremely complex, and requires a number of different techniques to be applied to overcome this issue. Furthermore, accidents are likely to occur during the actual development of the software, and this is another main reason why the situation remains unchanged.

One of the hardest parts of building software is in regard to the actual essence of the software, namely the specification, design, and testing of the conceptual constructs, given the complexity required in modern software. There are a number of difficulties that will often be presented in the actual software itself, such as when creating complex conceptual structures such as data sets. Furthermore, the relationships among data items, as well as the actual algorithms that make up the software, can have a direct influence on the complexity level of the software.

Difficulties can also occur relating to the production of software, as software is usually built by a team, communication is a key aspect related to the software development, and this can often lead to difficulties in communication, this may cause unreliability, and a difficulty of extending programs without creating unwanted side effects, which can lead to accidents occurring. An example of this is during the development of software, where a bug may be introduced by a developer unintentionally.

Task 1(b)

Explain in what ways Extreme Programming acts on those reasons/causes, resulting in better software-development outcomes.

There are a number of ways that Extreme Programming can act on the previously stated reasons, which will result in better software-development outcomes, such as customer involvement, iteration and release planning, code ownership, clean code and test-driven development.

Firstly, involving the customer with the development process, such as in release planning, developing user stories, and by undertaking iteration demos, will ensure that the software satisfies the customer needs. In turn, any misunderstanding between the customers and developers is less likely to occur. Furthermore, short development cycles will result in early, concrete, and continuing feedback.

Iteration and release planning that takes place at the last responsible moment will ensure that planning is done when the team has the most information from the customer, which means that accidents relating to the implementation of the software are less likely to occur. The ability to flexibly schedule the implementation of functionality means that the project can easily respond to changing business needs which will reduce the risks of project failure.

The collective ownership of the team's code, as well as pair programming, will make sure that all of the team members have some level of exposure to the code, making misunderstandings less likely. Furthermore, if a developer was to leave the project, then it is highly likely that another member of the team will be familiar with the code that has already been written. In addition, maintaining clean and well-structured code will make the project much more understandable, and in turn easier to extend, which will reduce the chances of any bugs occurring and accidents happening.

The Test-Driven Development requires a smaller number of steps to achieve the target functionality, client feedback can also be gathered at this stage resulting in changes being easier to make. Having a reliance on automated tests written by the programmers, customers and testers will ensure that the system is able to evolve, and any defects can be identified and addressed as soon as possible.

Task 2(a)

Extreme programming insists on the software being in a deployable state at all times (that is: at the end of every iteration the software can be demoed to the clients). What makes this possible?

Extreme Programming makes use of the Agile development methodology. Agile priorities a number of key elements, including responding to change instead of following a set plan, as well as developing software over writing detailed documentation.

The software that has been developed is built in order to meet the set needs of the customer in a timely fashion, this is achieved by delivering updates to the software frequently, as Agile has a much shorter timescale compared with other development methodologies such as the waterfall approach.

The most commonly used method to validate the success of a project is by undertaking a timely evaluation of the working software. An approach that utilises extreme programming will usually have very short development cycles, in turn this means that during every iteration of the development process, the team will have given much consideration to the various stages required for the project, such as the planning, analysis, design, and actual coding of the project.

The testing, implementation and evaluation stages will also be completed during the iteration, with direct interaction with the client, whereas the waterfall approach is much more time consuming as stages are constantly revised and revisited. As Extreme Programming makes use of an Agile approach, the work is completed in sprints of and the project will be ready to deploy at the end of each sprint, which will usually be completed as either a week or two-week sprints.

Task 2(b)

What role do customers/clients play in extreme programming?

During the development of the project, the customer is able to dictate to the developers exactly what it is that they want, and therefore any features can be implemented, and changes can be quickly made at any stage during the development process, even if the feedback is given during a late stage of the iteration. In addition to this, an MVP can be quickly developed and the customer is able to visualise what the project will look like, and recommend any changes that are to be made, as often the customer does not know exactly what they want until a working version is presented to them.

The development team is also able to make any necessary changes to the project specification at any stage in the development process, this means that it is possible to refine the specification, and change features and functionality required based on project constraints or customer feedback. As previously mentioned, it is highly unlikely that the customer can specify the exact features and functionality that they want to be implemented before the MVP is demonstrated to them.

Whilst the developers are responsible for the implementation, the customer is responsible for defining the key features of the project, which will in turn define how the software behaves. If a feature has been implemented by a developer in different way than what is defined by the customer, confusion may occur regarding if the functionality is a bug or a feature. This is just one example of why communication between both the developer and customer is crucial to the project's success, as it is important for both parties to work together to avoid any misunderstanding between them.

Task 3

Your company, PneuFood, aims to reach every household and provide near-instant food deliveries from supermarkets via a network of pneumatic tubes. Pneumatic tubes are systems that propel cylindrical containers through networks of pipes by compressed air or by partial vacuum. They are used for transporting solid objects, as opposed to conventional pipelines, which transport fluids.

The company has built an initial network of pneumatic tubes with all of the necessary hardware to deliver food cylinders from any supermarket to any household, and now needs a software system that will allow users to operate the system. End users (households) pay fees to PneuFood to use the service. So, they are the company's clients and the software will need to satisfy their needs. [NOTE: Of course, also supermarkets are users of the system. However, you are asked to ignore this fact in your answers to this question.]

You are part of the extreme programming team that has been asked to develop the software system. End users will interact with the system through a web interface.

You are asked to sketch a release plan for the software Given the specific nature of the project you can act both as a programmer and as a client.

In your release plan please specify:

Task 3(a)

A list of stories identified for the release which capture key needs for the customers.

User Stories:

[1] **As a** *Customer*, **I want to** "be able to check for the availability of a specific product."
[2] **As a** *Customer*, **I want to** "be able to check for the expiry date of a specific product."
[3] **As a** *Customer*, **I want to** "be able to search for, and order any items that I desire."
[4] **As a** *Customer*, **I want to** "be able to review my order and check for delivery costs."
[5] **As a** *Customer*, **I want to** "be able to report any faults that may occur in my order."

[6] **As a** *Customer*, **I want to** "be able to place orders automatically for when stock becomes available."
[7] **As a** *Customer*, **I want to** "be able to view my order before I purchase so that I can review the order."
[8] **As a** *Customer*, **I want to** "be able to view all previous orders so that I order the same products."
[9] **As a** *Customer*, **I want to** "be able to perform price comparisons of products with different supermarkets (price match guarantee) and choose the cheapest product where applicable."
[10] **As a** *Customer*, **I want to** "be able to apply any filters to my product searches, such as price, type (organic, free-range etc.) and brand."
[11] **As a** *Customer*, **I want to** "be shown recommendations for similar products to what I'm currently viewing or what I have previously ordered."

[12] **As a** *Customer*, **I want to** "be able to choose a specific time slot for the delivery."
[13] **As a** *Customer*, **I want to** "be able to track my delivery after I have placed an order."

Question 3(b)

A priority for each story in terms of value for the clients using a point system where 3 = very valuable, 2 = valuable, 1 = optional.

The user stories listed in Question 3(a) are sorted by priority:

- User stories **[1] – [5]** have a priority of **3**.
- User stories **[6] – [11]** have a priority of **2**.
- User stories **[12] – [13]** have a priority of **1**.

Task 3(c)

A list of the stories that should be implemented in the first iteration.

User stories **[1] – [5]** should be implemented in the first iteration, as these have the highest priority.

Task 4

You have been asked to continue to develop a simple Calculator class, taking over from another programmer. Overleaf you will find two implementations of the class, one in Java and one in Python, together with unit tests for them.

The Calculator class can currently do two operations – addition and multiplication – on any two integer numbers. The result of such operations is stored in an accumulator register, R1, which is initially 0 (when an instance of calculator is created), overwriting any value previously stored in R1.

*Users of the class call the method evaluate with a string argument representing the operation they want to execute. For instance, evaluate("10 + 5") will result in the value 15 being stored in R1, while evaluate("5 * 5") will result in 25 being stored in R1. Users can then retrieve the last result of an operation by calling the method get_R1.*

Task 4(a)

While the production code (the class Calculator) written by the programmer so far is reasonably good, it violates some principles of clean code, which is essential for Extreme Programming. Choose one implementation of Calculator, study it, and indicate which refactorings would be needed to make the code cleaner (use line numbers to indicate which elements of the code should be affected). Please, ignore the test class for this answer.

In the Java implementation of Calculator, some refactoring is necessary to make the code cleaner:

- The names of some methods are too ambiguous, for example on line **17** the function named '*evaluate'* could be named better, such as '*evaluateExperession'*. Furthermore, on line **27** the method named '*parse'* should be named something else, as this could be confused with the Parse function that is built into Java.

- The variables declared on lines **19** and **20** have names that are not self-explanatory. These variables should be renamed to '*firstNumber'* and '*secondNumber'*.

- Some of the comments are unnecessary or should be changed, on line **33** the example at the end of the comment could be removed. On line **35** there is a duplicate comment, and this could simply be removed altogether.

- In regard to the methods, the '*findOperator'* method defined on lines **39** to **46** should be declared at the beginning of the class as the main method should come first. Furthermore, the '*add*' method (lines **9** and **10**), and '*multiply*' method (lines **13** and **14**), are unnecessary. Instead this arithmetic operation can be performed where the methods are called on line **22** and **24** respectively (line **22** should be 'R1 = a + b', line **24** should be 'R = a * b').

Task 4(b)

Write one or two tests which will force you to extend Calculator to add the following functionality: (1) the calculator will have a second register, R2, and (2) it will accept a new assignment instruction of the form Register = Value, which can set the value of either R1 or R2. Thus, for example, evaluate("R1 = 10") will result in the value 10 being stored in R1, while evaluate("R2 = 5") will result in 5 being stored in R2.

Two tests are required in order to include the desired functionality:

Test 1:

```
@Test
public void calculatorCanEvaluateR1() {
        calculator.evaluate("R1 = 10")
        assert.equals(10, calculator.getR1());
}
```

Test 2:

```
@Test
public void calculatorCanEvaluateR2() {
        calculator.evaluate("R2 = 5")
        assert.equals(5, calculator.getR2());
}
```

Task 4(c)

Suggest which changes to the code would be required for the implementation of Calculator to acquire the functionality and pass the test(s) mentioned above (use line numbers to indicate the elements of the code that would be affected).

In order for the implementation of the Calculator to acquire the functionality required to pass the tests detailed in part (b), a number of changes would need to be made to the program:

Firstly, a variable for **R2** would have to be declared on line **3**:

```
private int R2 = 0;
```

Then, a new method would have to be added at line **9** to get the value of **R2**:

```
int getR2() {
        return R2;
}
```

Lastly, the add and multiply functions need an additional line for **R2** on lines **11** and **15**:

```
void add(int a, int b) {
        R1 = a + b;
        R2 = a + b;
}

void multiply(int a, int b) {
        R1 = a * b;
        R2 = a * b;
}
```

calculator.py

```
class Calculator:
    OPERATORS = ['+', '*']

    def __init__(self):
        self.R1 = 0

    def get_R1(self):
        return self.R1

    def add(self, a, b):
        self.R1 = a + b

    def multiply(self, a, b):
        self.R1 = a * b

    def evaluate(self, expression):
        elements = self.parse(expression)
        a = int(elements[0])
        b = int(elements[2])
        if elements[1] == "+":
            self.add(a, b)
        else:
            self.multiply(a, b)

    def parse(self, expression):
        position = self.find_operator(expression)
        return [
            # This is the first operand
            # (strip removes leading and trailing spaces)
            expression[:position].strip(),
            # This is the operator (e.g., *, /, -, ...)
            expression[position],
            # This is the second operand
            expression[position + 1:].strip()]

    def find_operator(self, expression):
        for operator in self.OPERATORS:
            operator_position = expression.find(operator)
            if operator_position != -1:
                return operator_position
        return -1
```

calculator_test.py

```python
import unittest
from calculator import Calculator


class CalculatorTest(unittest.TestCase):

    def setUp(self):
        self.calc = Calculator()

    def test_calculator_initial_value_of_register_should_be_zero(self):
        self.assertEqual(0, self.calc.get_R1())

    def test_calculator_can_add(self):
        self.calc.add(10, 20)
        self.assertEqual(30, self.calc.get_R1())

    def test_calculator_can_multiply(self):
        self.calc.multiply(10, 20)
        self.assertEqual(200, self.calc.get_R1())

    def test_calculator_can_parse_expression(self):
        parts = self.calc.parse("10 + 20")
        self.assertEqual("10", parts[0])
        self.assertEqual("+", parts[1])
        self.assertEqual("20", parts[2])

    def test_calculator_can_evaluate_expression(self):
        self.calc.evaluate("10 * 20")
        self.assertEqual(200, self.calc.get_R1())
```

Calculator.java

```java
public class Calculator {
   private int R1 = 0;
   private final char[] OPERATORS = {'+','*'};

   int getR1() {
      return R1;
   }

   void add(int a, int b) {
      R1 = a + b;
   }

   void multiply(int a, int b) {
      R1 = a * b;
   }

   public void evaluate(String expression) {
      String[] elements = parse(expression);
      int a = Integer.parseInt(elements[0]);
      int b = Integer.parseInt(elements[2]);
      if (elements[1].equals("+"))
         add(a, b);
      else
         multiply(a, b);
   }

   String[] parse(String expression) {
      int position = findOperator(expression);
      return new String[]{
               // This is the first operand
               // (trim removes leading and trailing spaces)
               expression.substring(0,position).trim(),
               // This is the operator (e.g., *, /, -, ...)
               expression.substring(position,position+1),
               // This is the second operand
               expression.substring(position+1).trim()};
   }

   private int findOperator(String expression) {
      for(char operator: OPERATORS) {
         int operatorPosition = expression.indexOf(operator);
         if ( operatorPosition != -1)
            return operatorPosition;
      }
      return -1;
   }
}
```

CalculatorTest.java

```java
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

public class CalculatorTest {

   private Calculator calculator;

   @Before
   public void setUp() {
      calculator = new Calculator();
   }

   @Test
   public void calculatorRegisterShouldInitiallyBeZero() {
      assertEquals(0,calculator.getR1());
   }

   @Test
   public void calculatorCanAdd() {
      calculator.add(10,20);
      assertEquals(30,calculator.getR1());
   }

   @Test
   public void calculatorCanMultiply() {
      calculator.multiply(10,20);
      assertEquals(200,calculator.getR1());
   }

   @Test
   public void calculatorCanParseExpression() {
      String [] parts = calculator.parse("10 + 20");
      assertEquals("10",parts[0]);
      assertEquals("+",parts[1]);
      assertEquals("20",parts[2]);
   }

   @Test
   public void calculatorCanEvaluateExpression() {
      calculator.evaluate("10 * 20");
      assertEquals(200, calculator.getR1());
   }
}
```