**Table of Contents**

1. Risk, quality and communication management…………………………….…..3
   1. Risk management……………………………………………………………………...…3
   2. Quality management…………………………………………………..…….……..….5
   3. Communication management………………………………………………………5
2. Development methodology………………………………………………………………7
3. Java implementation of “Place Order” use case………………………….…….9
   1. Design Class Diagram……………………………………………………………………9
   2. Java classes for “Place Order”…………………………………………………….10
4. Artefacts of agile Scrum methodology ……………………………..……….……14
   1. User stories………………………………………………………………………..………14
   2. Backlogs……………………………………………………………………………………..16
   3. Burndown charts………………………………………………………………….…….22
5. Testing…………………………………………………………………………………….………24
   1. Test methodology……………………………………………………………………...24
   2. Test cases and descriptions of tests……………………………………………25
6. References……………………………………………………………………………………..27
7. **Risk, quality and communication management – Takeaway Restaurant Project**
   1. **Risk management**

Risk can be defined as an uncertain event which may occur in the future and can delay or prevent the achievement of the project’s goal.   
In order to manage risk in our project first we need to identify, assess and prioritize risks, then we need to develop a plan to prevent risks and limit their impact on the project.  
The most important document of risk management is the Risk Register.  
**The Risk Register** highlights the most likely threats to a software project with regards to each user story.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID, Corresponding User story ID** | **Date** | **Risk Description** | **Likelihood** | **Impact** | **Owner** | **Action** |
| R1, S01 | 01 April | Browse menu page is not displayed fast enough causing frustration to customer | Medium | High | Scrum Master | Ensure enough server resources are provided |
| R2, S02 | 02 April | Registration is process overly complicated, discourages user from proceeding | Medium | High | Scrum Master | Ensure that registration form is easy to use |
| R3, S03 | 02 April | Login process too slow, discourages user from proceeding | Medium | High | Scrum Master | Ensure that login form is easy to use |
| R4, S04 | 03 April | Structure of user account page is too complicated and difficult to update | Medium | Medium | Scrum Master | Ensure clear and concise structure of user account page |
| R5, S05 | 04 April | Content of shopping basked is not displayed correctly causing confusion to customer | Medium | High | Scrum team | Ensure correct functionality of page components by testing |
| R6, S06 | 05 April | Order status is not displayed correctly | Medium | Medium | Scrum team | Ensure fast and correct status display through testing |
| R7, S07 | 06 April | Payment options are not listed correctly | High | High | Scrum team | Excessive testing on all features regarding payment options |
| R8, S08 | 07 April | Cash payment option prompts customer to pay in advance | Low | Medium | Scrum team | Thoroughly test all payment options |
| R9, S09 | 08 April | Card payment authorization process is too slow | Medium | High | Scrum master | Provide enough resources to process online payments fast |
| R10, S10 | 09 April | System does not process discount codes correctly | High | High | Scrum master | Thoroughly test code that processes discount codes/vouchers |
| R11, S11 | 10 April | Customer orders are not displayed correctly to staff members | Medium | High | Scrum team | Test all system display functionality |
| R12, S12 | 11 April | Update order status functionality works too slow | Medium | Medium | Scrum team | Provide sufficient resources to ensure speedy system response |
| R13, S13 | 12 April | Manage menu functionality works too slow | Medium | Medium | Scrum team | Test add, delete, update menu functions |
| R14, S15 | 13 April | View transactions page updates slowly | Medium | High | Scrum team | Provide minimal amount of code and speedy execution through TDD |

* 1. **Quality management**

Project quality includes two main considerations: **conformance to requirements** outlined in the project specification and **fitness to use**.

In our project we wish to maintain high quality standards by continuously performing Software Quality Assurance and Quality Control.  
**Software Quality Assurance** ensures that work items are monitored and comply with the industry standards, such as ISO 9000.  
With regards to **Quality Control**, the most important activity is **testing**, which will be carried out in every phase of the development process.  
The incremental and iterative nature of our chosen development methodology, agile, will help ensure that testing is done on each and every software component.

Testing process must cover the testing of each software component (**unit testing**); **integration testing, system testing and user acceptance testing**.

* 1. **Communication management**

Communication management plays a crucial role in software project management.  
It ensures that all relevant project information is generated, distributed and stored appropriately.   
Our communication strategy towards the stakeholders is detailed in the **Communication Management Plan**.  
The Communication Management Plan is the responsibility of the project manager and outlines the information needs of the stakeholders, the methods for distributing information and contains the frequency and expected outcomes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Stakeholder** | **Information to convey** | **Speaker** | **Tool to convey information** | **Frequency** | **Expected outcome** |
| Project team | Project updates | Project manager | Project communication plan | Monthly | All deadline are within the expectations |
| Project manager | Proposed project changes | Senior company management | Emails | As often as changes require | Change requests implemented within reasonable timeframe |
| Customer | Project status updates | Project manager | Face-to-face meeting, reports, presentations | Monthly | Customer feedback and change requests |

Communication plays a major role in agile software development methodology. Agile Scrum framework emphasises the importance of **face-to-face interaction** over written methods of communication in the form of **Project Kickoff Meeting, Sprint Planning Meetings and Daily Scrum Meetings.**We would utilize all the mentioned information distribution methods in order to enhance productivity and ensure that changes are implemented with minimum delay.

1. **Development methodology**

In our project we would like to employ the principles of the agile software development methodology.   
***Agile development*** is software development methodology that responds to change and unpredictability through ***iterative and incremental*** work cadences. It consists of 5 basic steps: **analysis, design, development, testing and release.** These steps are repeated iteratively during the development of a software product in order to best respond to change.   
The process is also incremental in nature as new features are added in each iteration, resulting in a potentially working piece of software.  
As client requirements may change at any stage, development process must be fast and flexible.  
Agile development has several well-defined principles.  
The most important one is ***customer satisfaction***, where early and continuous delivery of working software is the main goal. This brings the advantage of early feedback from the client and increases the quality of the final product.  
Agile as a methodology also ***welcomes changes* at any stage** of the development process. Change may bring competitive advantage for the client, as it helps understand market requirements.   
In order to best accommodate changing requirements, software development teams keep **software structure as simple and flexible as possible** to ensure that the impact of change remains minimal.  
**Working *software product is delivered as early as possible*** in regular intervals during the development process. This practice ensures that the client can provide feedback early and changes can be accommodated immediately.  
Agile processes emphasize the importance **of *face-to-face communication*** over written methods of information exchange.  
Other core agile principles include ***sustainable development*** where the development team paces itself to maintain the highest quality coding standards; ***self-organizing teams***, where responsibilities are communicated to the team as whole and team members decide how to best fulfil these requirements; and ***regular reflection on effectiveness***, where the team regularly adjusts its organisation and methods in order to meet the requirements.

Agile development is an umbrella term that has several well-developed subsets.   
The most commonly used agile process frameworks are Extreme Programming, Feature-Driven Development, Kanban and Scrum.

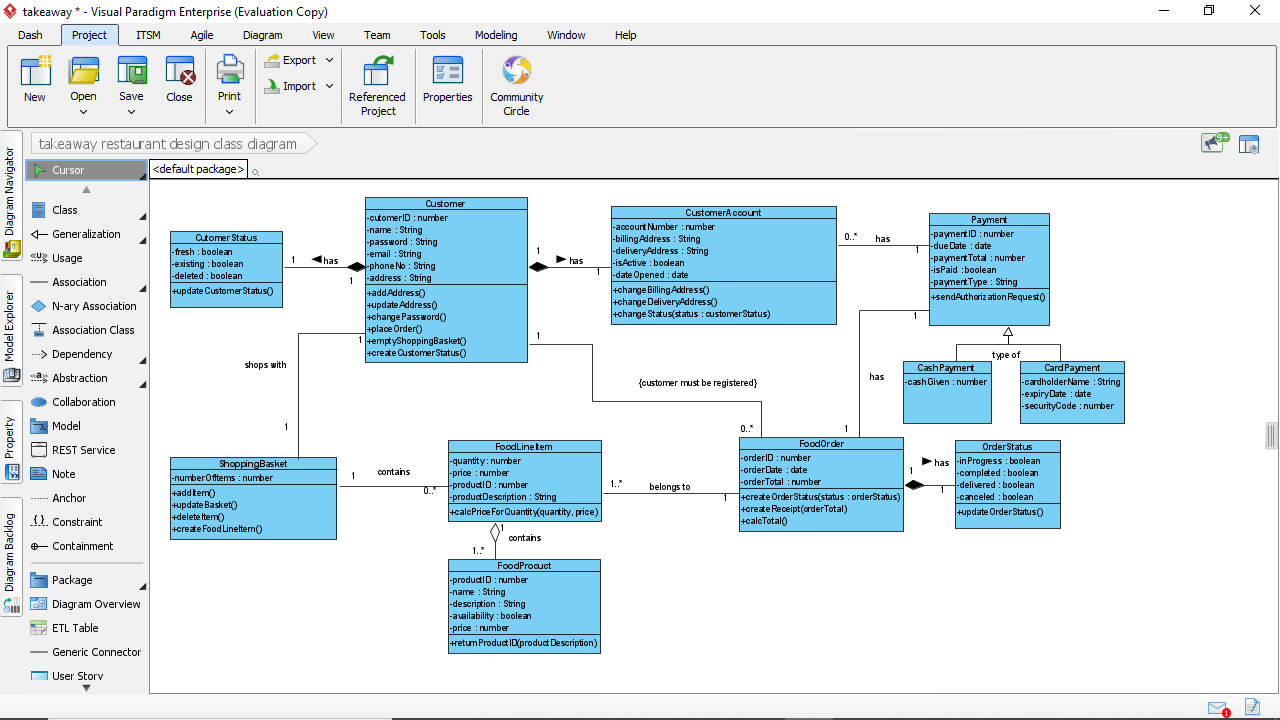
As our project is geared towards the development of an online food ordering system for a takeaway restaurant, it is highly likely that client requirements would change during the development process.  
In order to best accommodate changing client requirements; in our project we will follow the main practices of ***Scrum methodology***.  
Scrum is lightweight, easy-to-adopt agile framework where work is organized into smaller, manageable pieces, completed by cross-functional teams in a given timeframe. The timeframes are called ***sprints*,** with a typical duration of 2-4 weeks.  
Scrum process is based on the production of several important **artefacts**, including the Product Backlog, sprint backlogs, user stories and progress burndown charts.  
***Product Backlog*** is the most important document of the development process, containing all the required functionality of the proposed software system.   
As the master list of all features, Product Backlog outlines requirements in a form of a prioritized list of **Product Backlog Items (PBIs).**  
***Sprint backlogs*** are based on the content of the Product Backlog; PBIs that the team wants to complete during a given sprint are selected and broken down into individual tasks in this document.

***User stories*** are the high-level definitions of customer requirement, written from the client’s point of view. These stories must provide enough information for the development team to produce an effort estimate to implement them.  
User stories are usually assigned with an indication of priority (high, medium, low), a unique story ID and a story point. These story points are then used to estimate the effort required for the implementation of a given story.  
***Burndown charts*** are the graphical representation of progress and remaining work during a sprint. These charts are produced daily in order to monitor progress and provide transparency to the team.

1. **Java implementation of “Place Order” use case**
   1. **Design Class Diagram**

Design class diagram is usually created after the completion of interaction diagrams during the object oriented process of software engineering.   
Design class diagrams are based on the conceptual class diagram, but   
while conceptual class diagrams illustrate real world concepts and contain attributes of the classes, design class diagrams contains **software components** and in addition to **attributes**, we include **operations** (methods) associated with the behaviour of each class.

Naturally, associations, navigability, generalisation and inheritance between classes are shown in the design class diagram as well.



* 1. **Java classes for “Place Order”**

Our chosen CASE tool, Visual Paradigm offers a functionality to turn design class diagram into computer code, using either Java or C++ programming languages.   
Based on our previous studies of the Java programming language, we generated the following Java classes from the design class diagram with regards to the most important use case, Place Order:

public class **Customer** {

private int customerID;

private String name;

private String password;

private String email;

private String phoneNo;

private String address;

public void addAddress() {

// TODO - implement Customer.addAddress

}

public void updateAddress() {

// TODO - implement Customer.updateAddress

}

public void changePassword() {

// TODO - implement Customer.changePassword

}

public void placeOrder() {

// TODO - implement Customer.placeOrder

}

public void emptyShoppingBasket() {

// TODO - implement Customer.emptyShoppingBasket

}

public void createCustomerStatus() {

// TODO - implement Customer.createCustomerStatus

}

}  
public class **ShoppingBasket** {

private int numberOfItems;

public void addItem() {

// TODO - implement ShoppingBasket.addItem

}

public void updateBasket() {

// TODO - implement ShoppingBasket.updateBasket

}

public void deleteItem() {

// TODO - implement ShoppingBasket.deleteItem

}

public void createFoodLineItem() {

// TODO - implement ShoppingBasket.createFoodLineItem

}

}

public class **FoodLineItem** {

private int quantity;

private double price;

private int productID;

private String productDescription;

public void calcPriceForQuantity() {

// TODO - implement FoodLineItem.calcPriceForQuantity

}

}

public class **FoodProduct** {

private int productID;

private String name;

private String description;

private boolean availability;

private double price;

public void returnProductID() {

// TODO - implement FoodProcuct.returnProductID

}

}

public class **FoodOrder** {

private int orderID;

private date orderDate;

private double orderTotal;

public void createOrderStatus(orderStatus status) {

// TODO - implement FoodOrder.createOrderStatus

}

public void createReceipt() {

// TODO - implement FoodOrder.createReceipt

}

public void calcTotal() {

// TODO - implement FoodOrder.calcTotal

}

}

public class **OrderStatus** {

private boolean inProgress;

private boolean completed;

private boolean delivered;

private boolean canceled;

public void updateOrderStatus() {

// TODO - implement OrderStatus.updateOrderStatus

}

}

public class **Payment** {

private int paymentID;

private date dueDate;

private double paymentTotal;

private boolean isPaid;

private String paymentType;

public void sendAuthorizationRequest() {

// TODO - implement Payment.sendAuthorizationRequest

}

}

1. **Artefacts of agile scrum methodology**
   1. **User stories**

User stories are high level definitions of client requirements and they serve as building blocks in the Product Backlog. These stories are written from the user’s point of view and represent the lowest level of functional decomposition.   
Typically a user story is written in a **role-functionality-benefit format**, where the role represents an actual human who interacts with the system, functionality describes the action the user wants to take and benefit outlines a result the user wishes to achieve through the action.  
User stories usually have a story ID, a level of importance (priority) and story points assigned to them. Story points will be used the create effort estimates during sprints.

|  |  |  |  |
| --- | --- | --- | --- |
| **Role, functionality, benefit** | **Story ID** | **Priority** | **Story points** |
| As a user I want to browse the menu of the restaurant so that I can choose my food | S01 | High | 4 |
| As a user I want to register as a customer so that I can place a food order | S02 | High | 4 |
| As a user I want to login, so that I can view my orders | S03 | High | 6 |
| As a user I want to see my user account so that I can update my details | S04 | High | 6 |
| As a user I want to see the content of my shopping basket so that I can update it | S05 | High | 5 |
| As a user, I want to check the status of my order so that I know if it’s ready | S06 | Medium | 2 |
| As a user I want to see payment options so that I can choose one | S07 | High | 4 |
| As a user I want to have cash payment option on collection so that I don’t have to pay in advance | S08 | High | 2 |
| As user I want to pay for my order online using a credit/debit card so that I can process the entire order online | S09 | High | 6 |
| As a user I want to use promotional codes to get discount from the price | S10 | Medium | 2 |
| As a staff member I want to view the orders the users placed | S011 | High | 3 |
| As a staff member I want to update the status of orders | S12 | Medium | 4 |
| As a system admin I want to manage the menu so that I can add, change and delete a food item | S13 | High | 4 |
| As a system admin I want to manage customers so that I can add, update or delete a customers’ details | S14 | High | 8 |
| As a system admin I want to view transactions to that I can see which orders have been paid for | S15 | High | 6 |

* 1. **Backlogs**

Agile scrum methodology uses 2 basic types of backlogs: **Product Backlog** and **sprint backlogs**.   
The Product Backlog is an overall, prioritized list of all features a proposed software system needs to include, in a form of Product Backlog Items (PBIs).   
PBIs are often written in a form of a user story.  
This document can include all functional, non-functional and technical requirements as well as knowledge acquisition methods.  
The Product Backlog is created during the project kickoff meeting and is the responsibility of the product owner and the scrum master.  
Sprint backlogs outline the PBIs that the development team wants to complete during a given sprint. PBIs are usually broken down into smaller tasks (sprint tasks).   
Sprints tasks specify a unit of work to be completed by one team member and typically represent a workload of 4-16 hours.  
Sprints backlogs are created during the sprint planning meetings by the product owner, scrum master and team members.

**Product Backlog**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Priority** | **PBI** | **Related user story** | **Backlog ID** | **Story points** | **Effort estimate (\*2.5 hours)** |
| 1 | Create Database | As a system admin I want to manage customers so that I can add, update or delete customers’ details | S14 | 8 | 20 |
| 2 | Create menu page | As a user I want to browse the menu of the restaurant so that I can choose my food | S01 | 4 | 10 |
| 3 | Create registration page | As a user I want to register as a customer so that I can place a food order | S02 | 4 | 10 |
| 4 | Create login page | As a user I want to login, so that I can view my orders | S03 | 6 | 15 |
| 5 | Create User Accounts page | As a user I want to see my user account so that I can update my details | S04 | 6 | 15 |
| 6 | Create Shopping Basket | As a user I want to see the content of my shopping basket so that I can update it | S05 | 5 | 12.5 |
| 7 | Create order status | As a user, I want to check the status of my order so that I know if it’s ready | S06 | 2 | 5 |
| 8 | Create payment options | As a user I want to see payment options so that I can choose one | S07 | 4 | 10 |
| 9 | Create cash payment | As a user I want to have cash payment option on collection so that I don’t have to pay in advance | S08 | 2 | 5 |
| 10 | Create card payment | As user I want to pay for my order online using a credit/debit card so that I can process the entire order online | S09 | 6 | 15 |
| 11 | Create redeem voucher | As a user I want to use promotional code so that I can get discount from the price | S10 | 2 | 5 |
| 12 | Create view orders | As a staff member I want to view the orders the users placed so that I can complete them | S11 | 3 | 7.5 |
| 13 | Create manage order status | As a staff member I want to update the status of orders so that I can inform customers of order status | S12 | 4 | 10 |
| 14 | Create manage menu | As a system admin I want to manage the menu so that I can add, change and delete a food item | S13 | 4 | 10 |
| 15 | Create view transactions | As a system admin I want to view transactions so that I can see which orders have been paid for | S15 | 5 | 10 |

**Sprint backlog 1**

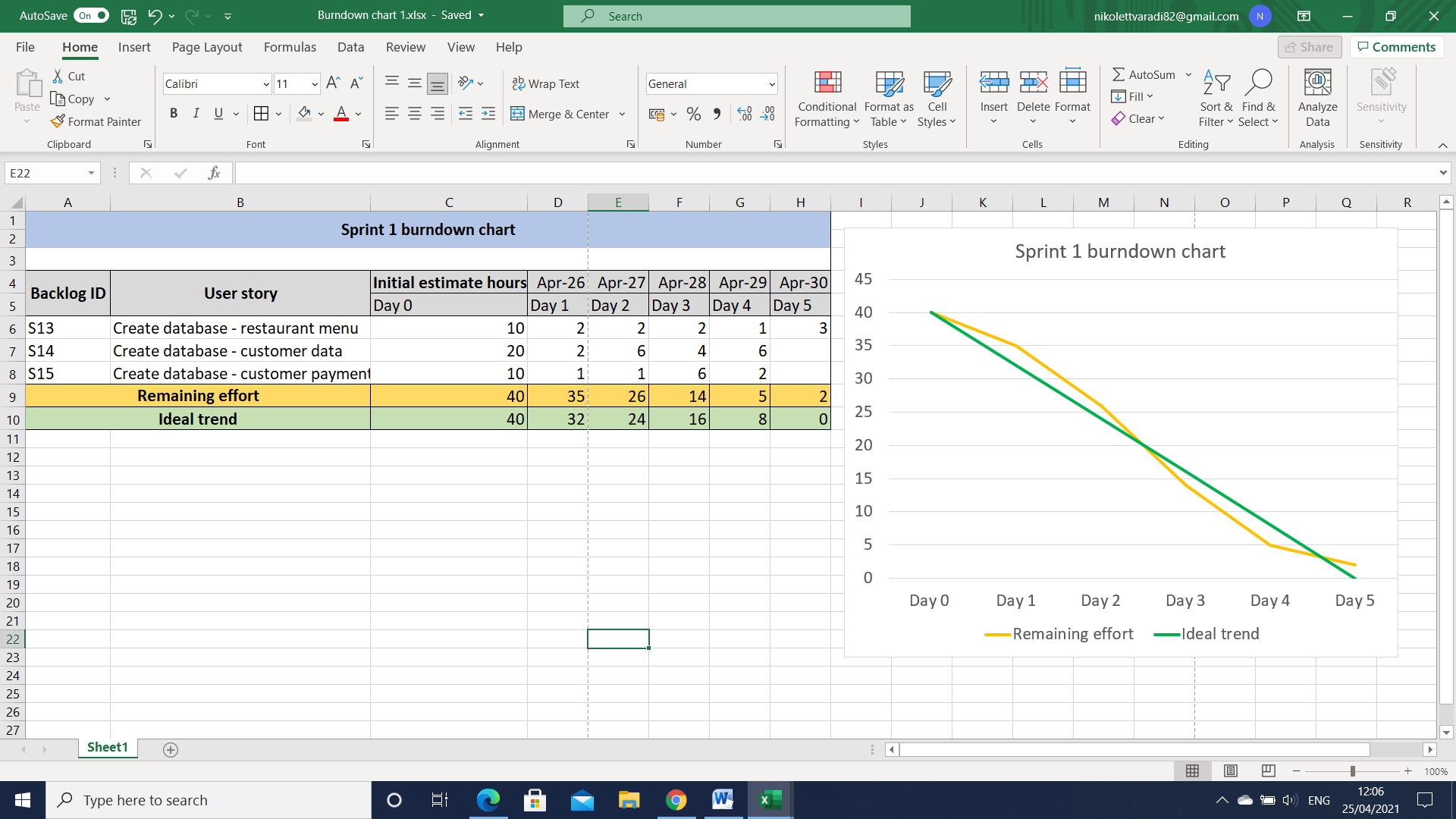
|  |  |  |  |
| --- | --- | --- | --- |
| **Priority** | **PBI and task breakdown** | **Backlog ID** | **Effort estimate**  **(hours)** |
| 1 | **Create Database** | S14 | 20 |
|  | Design CustomerData table |  | 5 |
|  | Design CustomerOrders table |  | 5 |
|  | Create CustomerData table |  | 5 |
|  | Create CustomerOrders table |  | 5 |
| 2 | **Create Database** | S13 | 10 |
|  | Design RestaurantMenu table |  | 5 |
|  | Create ResturantMenu table |  | 5 |
| 3 | **Create Database** | S15 | 10 |
|  | Design CustomerPayments table |  | 5 |
|  | Create CustomerPayments table |  | 5 |

**Sprint backlog 2**

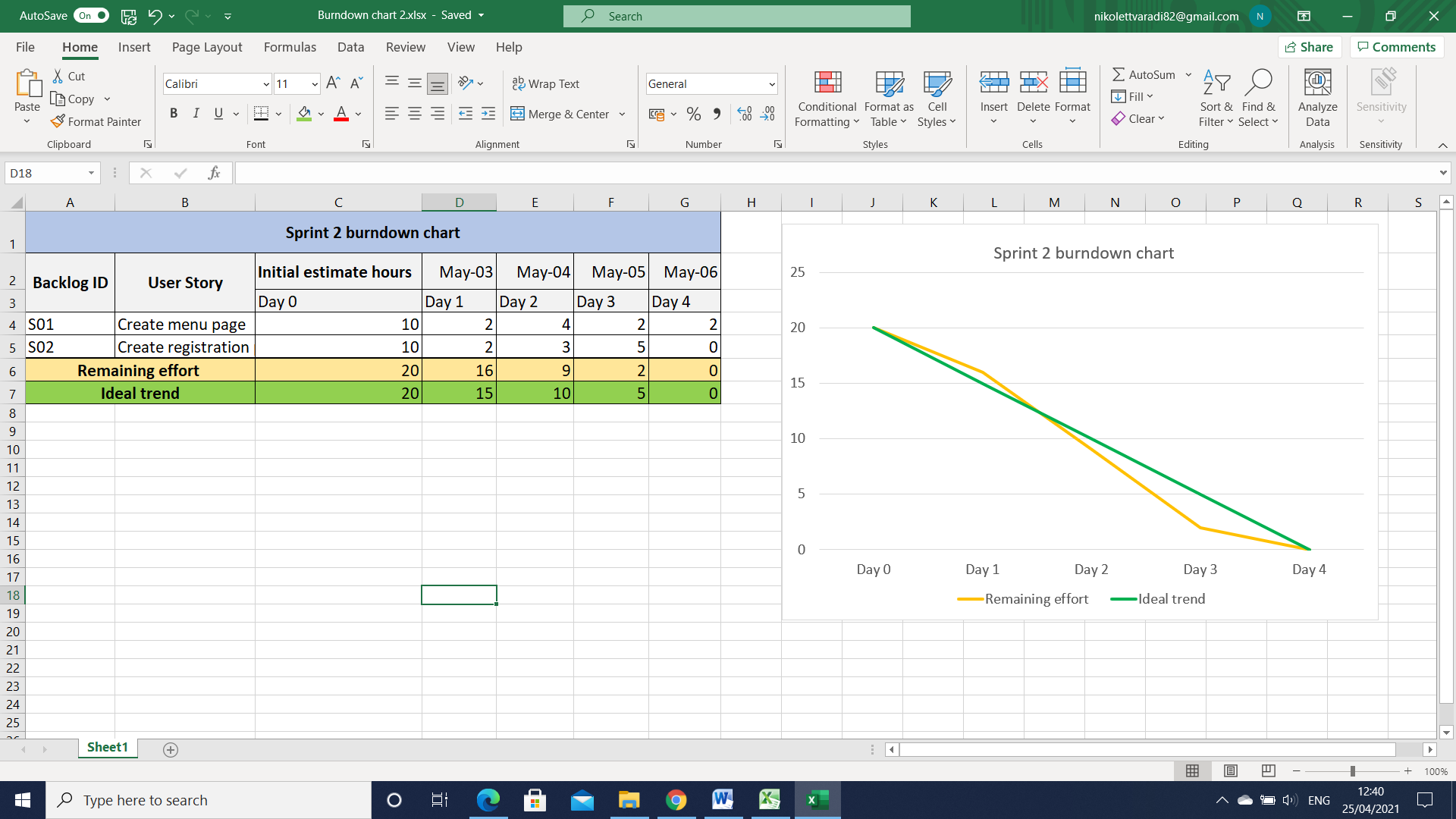
|  |  |  |  |
| --- | --- | --- | --- |
| **Priority** | **PBI and task breakdown** | **Backlog ID** | **Effort estimate (hours)** |
| 4 | **Create menu page** | S01 | 10 |
|  | Develop HTML |  | 2 |
|  | Create CSS styling |  | 3 |
|  | Write JavaScript code for functionality |  | 5 |
| 5 | **Create registration page** | S02 | 10 |
|  | Create HTML |  | 1 |
|  | Create Registration Form Components |  | 2 |
|  | Create CSS styling |  | 3 |
|  | Write JavaScript code for functionality |  | 4 |

* 1. **Burndown charts**

The first burndown chart shows the progress of database creation, 40 hours of workload, during 5 working days:



The second burndown chart illustrates the creation of restaurant menu and registration page development, 20 hours workload in 4 days:



1. **Testing**
   1. **Testing methodology**

Testing is the process of running software in order to find errors and can only show the presence of faults.

There are two basic types of testing, component testing and system testing.  
***Component testing*** (unit testing) is a process carried out by the developers themselves based on their experience on individual program components.  
*System testing* on the other hand is the testing of groups of components, integrated into a subsystem, usually carried out by a team of independent testers.  
Our goals during the testing process could be either **validation *testing***, where we demonstrate that the system meets its requirements; or ***defect testing****,* where the aim is to discover faults and the presence of bugs.

With regards to our project we wish to carry out component testing and defect testing so as to discover program anomalies and bugs.

Our chosen testing methodology is **Test Driven Development**.   
This kind of testing is often associated with the practices of Extreme Programming.  
In Test Driven Development tests are written prior to coding and just enough code is produced to pass the unit test.  
Ideally, testing should be comprehensive, covering all object classes, their attributes, methods and states.  
However, in practice exhaustive testing is not possible. Instead, testing policies highlight the most important features to be tested.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Scenario:  Check add address functionality, “Customer” class | | | | | | | |
| **Test case ID** | **Description** | **Precondition** | **Test data** | **Expected result** | **Post condition** | **Actual result** | **Status** |
| T01 | Enter a valid address | Customer must be logged in | <Valid address> | Address successfully added | Address added message shown | Address added successfully | Pass |
| T02 | Enter invalid address | Customer must be logged in | <Invalid address> | Address not added | Address invalid message shown | Address not added | Pass |
| Scenario:  Check empty shopping basket functionality, “Customer” class | | | | | | | |
| **Test case ID** | **Description** | **Precondition** | **Test data** | **Expected result** | **Post condition** | **Actual result** | **Status** |
| T03 | Click “Empty Basket” button | There must be items already added into the basket | <Items in basket> | Items deleted from shopping basket | Shopping basket empty message shown | Items deleted from shopping basket | Pass |
| T04 | Click “Empty Basket” button | Shopping basket is initially empty | <None> | No deletion carried out | Shopping basket empty message shown | No deletion, shopping basket empty message shown | Pass |
| Scenario:  Check add item functionality, “ShoppingBasket” class | | | | | | | |
| **Test case ID** | **Description** | **Precondition** | **Test data** | **Expected result** | **Post condition** | **Actual result** | **Status** |
| T05 | Click on “Add item” button under food item | There must be a food item chosen by customer | <Food item> | Item successfully added to basket | Item added to basket message shown | Item successfully added to basket | Pass |
| Scenario:  Check calculate price for quantity functionality, “FoodLineItem” class | | | | | | | |
| **Test case ID** | **Description** | **Precondition** | **Test data** | **Expected result** | **Post condition** | **Actual result** | **Status** |
| T06 | Click on calculate “Basket Total” button | There must be items in the basket | <Content of basket> | Grand total value of shopping basket calculated | Grand total displayed to customer | Grand total calculated, displayed | Pass |
|  |  |  |  |  |  |  |  |

* 1. **Test scenarios, test cases and description of tests**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Scenario:  Check send authorization request functionality, “Payment” class | | | | | | | |
| **Test case ID** | **Description** | **Precondition** | **Test data** | **Expected result** | **Post condition** | **Actual result** | **Status** |
| T07 | Click on “Process payment” button | Total for basket is successfully calculated | <Total amount to pay> | Card payment processed successfully | “Payment successful” message displayed | Payment processes, appropriate message displayed | Pass |
| Scenario:  Check create order status functionality, “FoodOrder” class | | | | | | | |
| **Test case ID** | **Description** | **Precondition** | **Test data** | **Expected result** | **Post condition** | **Actual result** | **Status** |
| T08 | Click on “Order status” button | Order is successfully paid for | <Payment confirmation message> | Status of order is created and displayed | Appropriate message is displayed to customer: “In progress” or “Completed” | No message displayed | Failed |
| Scenario:  Check update order status functionality, “OrderStatus” class | | | | | | | |
| **Test case ID** | **Description** | **Precondition** | **Test data** | **Expected result** | **Post condition** | **Actual result** | **Status** |
| T09 | Click on dropdown menu containing status options | Customer order is successfully received and paid for | <Status options> | Status of order successfully changed | “Order status changed successfully” message appears on screen | No message appears, no update carried out | Failed |

1. **References**

Dennis, A., Haley Wixom, B. and Tegarden, D., 2015. *Systems Analysis and Design with UML*. 5th ed. New York, US: John Wiley & Sons, Inc.

Visual Paradigm. 2021. Visual Paradigm. [ONLINE] Available at: <https://www.visual-paradigm.com/>. [Accessed 30 March 2021].

Scrum Institute. 2021. Scrum Institute. [ONLINE] Available at: <https://www.scrum-institute.org/the-scrum-product-backlog-the-scrum-framework.php>. [Accessed 19 April 2021].

Agile Zone. 2021. DZone. [ONLINE] Available at: <https://dzone.com/articles/agile-best-practices-every-agile-team-should-have>. [Accessed 19 April 2021].