**CS673 Software Engineering** 

**Team 3 - Bug killer**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Siyuan Xue | Team Leader | *Siyuan Xue* | 11/10/2023 |
| Tony Xiang | Configuration Leader | *Qian Xiang* | 11/10/2023 |
| Xiankun Niu | Design and Implementation Leader | *Xiankun Niu* | 11/10/2023 |
| Zhuoru Song | Requirement Leader | *Zhuoru Song* | 11/10/2023 |
| Jialin Li | QA leader | *Jialin Li* | 11/10/2023 |
| Chandrasiri | Security leader | *Chandrasiri* | 11/10/2023 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **0** | **All member** | **9/26/2023** | **Initial Draft** |
| **1** | **All member** | **10/18/2023** | **Fix issues** |
| **2** | **All member** | **11/10/2023** | **Fix issues** |
|  |  |  |  |

[Overview](#_heading=h.gjdgxs)

[Related Work](#_heading=h.30j0zll)

[Proposed High level Requirements](#_heading=h.1fob9te)

[Management Plan](#_heading=h.3znysh7)

[Objectives and Priorities](#_heading=h.2et92p0)

[Risk Management (need to be updated constantly)](#_heading=h.tyjcwt)

[Timeline (need to be updated at the end of each iteration)](#_heading=h.3dy6vkm)

[Configuration Management Plan](#_heading=h.1t3h5sf)

[Tools](#_heading=h.4d34og8)

[Deployment Plan if applicable](#_heading=h.2s8eyo1)

[Quality Assurance Plan](#_heading=h.17dp8vu)

[Metrics](#_heading=h.3rdcrjn)

[Code Review Process](#_heading=h.26in1rg)

[Testing](#_heading=h.lnxbz9)

[Defect Management](#_heading=h.35nkun2)

[References](#_heading=h.1ksv4uv)

[Glossary](#_heading=h.44sinio)

# 

# Overview //Siyuan Xue

The proposed system is an “Online Restaurant Ordering System” designed to facilitate the ordering process for both customers and the restaurant staff. It is developed using Python Flask. It involves the utilization of QR codes assigned to each table, enabling users to access the restaurant’s menu and place orders. The system is envisioned to optimize the ordering process, reducing wait times and human errors, while offering real-time updates to the kitchen staff. The system is mainly divided into 4 modules: user, kitchen, waiter, and management.

User Model：

Homepage:

navigate to and from each page

Cartpage：

Display the added dishes and calculate the quantity, navigate to the confirmation page

Menupage：

Displays information about all dishes and can be added to shopping cart

Register Page：

Register personal information, email verification

Loginpage：

Allows users to log in to the main page

Confirmation page：

It is a return page for many pages. Maybe we will consider having multiple return pages in the future to deal with different situations.

Kitchen Model：

Kitchen display page

Display the dishes need to be cook by orders and tables, send the message to customer and server when a dish is finished

Waiter Model：

Server mainpage

1. Show the ongoing orders information(table, dishes, status)
2. Respond to the calls of the customers (Alert functions)
3. Jump to the Completed orders history page when click the Alert button
4. Receive alerts for new order notification

Completed orders history page

Display the completed orders in the system, so that the server can know the avenue situation and report to the boss.

Management Model：

Leave to be created in Iteration 3

Conclusion:

The proposed Online Restaurant Ordering System utilizing Python Flask and QR codes aims to enhance the dining experience for customers while streamlining the order management process for the restaurant staff. By integrating proven methods from related works, including real-time notifications, seamless payment options, intuitive user interface design, and efficient order processing mechanisms, the system is poised to offer a comprehensive solution to modern restaurant management challenges.

# 

# Related Work //Siyuan Xue

Online Restaurant Ordering System VS MenuDrive

The same:

Online menu and ordering:

Like MenuDrive, our system also provides online menu browsing and ordering services, allowing customers to easily select food and place their order.

Promotion and discount management:

MenuDrive allows restaurants to conveniently manage promotions and discounts to attract more customers. We allow restaurants to change food prices at will to achieve the same goal

Customer management:

Both our system and MenuDrive provide customer management capabilities, including order history and customer preferences.

The difference:

User interface and experience:

Our system has a more intuitive and user-friendly interface, using modern design and clear navigation to provide a better user experience.

Customization options:

Our system is completely owned by the restaurant, so the restaurant can change the system values at any time, and we provide paid change services at any time.

Real-time order updates:

Our system provides real-time order updates and notifications, ensuring restaurants are instantly aware of order status and progress.

Payment options:

Our system does not provide an online payment system. In fact, we will print a receipt directly. The customer only needs to show this receipt, and the restaurant can collect money offline based on the amount.

# Proposed High level Requirements //zhuoru Song

* 1. Functional Requirements  
     (For each functional requirement, please give a feature title and a brief description using the following format: As (a role), I want to (action), so that (value).)
     1. Essential Features (the core features that you definitely need to finish):

(For each essential features, please give a rough estimation in terms of person hours or an range of person hours)

**Display menu**

As a customer, I want to view the menu online, so that I can see what's available.

Person hours: 20

**Add items to cart**

As a customer, I want to add dishes to my cart, so that I can make an order later.

Person hours: 20

**Remove items from cart**

As a customer, I want to remove dishes from my cart, so that I can see the cart and exclude dishes that I no longer want.

Person hours: 5

**Make order**

As a customer, I want to make an order, so that the restaurant can start preparing my food and I can see a order page.

Person hours: 20

**Checkout**

As a customer, I want to checkout, so that the I can make a payment.

Person hours: 5

**Business login**

As a restaurant manager, I want to login to the system, so that I can manage it.

Person hours: 20

**Edit menu**

As a restaurant manager, I want to edit my restaurant menu, so that I can add or delete items from the menu.

Person hours: 20

**Manage table**

As a restaurant manager, I want to manage my restaurant tables, so that I can add or delete tables in my restaurant.

Person hours: 15

**Open/close business**

As a restaurant manager, I want to open and close my business, so that the system doesn’t allow new orders during close time.

Person hours: 5

* + 1. Desirable Features (the nice features that you really want to have too):

**Customer login**

As a member/VIP, I want to log in to the system, so that I can use my member benefit/discount and earn rewards.

Person hours: 15

**New User Registration**

As a new user, I want to register an account, so that I can log in as a member.

Person hours: 15

* + 1. Optional Features (additional cool features that you want to have if there is time):

**Notice to the kitchen**

As a kitchen staff, I want to get customer order information, so that I can prepare food as soon as possible.

Person hours: 10

* + 1. Existing Features (delete this item if your project starts from scratch)
  1. **Nonfunctional Requirements** //Chandrasiri
     1. **Security requirements**

Security requirements associated with our projects are as follows.

* **Sensitive data:** 
  + Sensitive data displayed to users should be masked, limiting access based on user roles.
  + All sensitive data stored in databases or transmitted over networks must be encrypted using industry-standard encryption algorithms.
  + Access to sensitive data or functionalities should be restricted to authorized users only, based on role-based access control.
* **Code Review and Security Testing:**
  + Conducting code reviews and security testing regularly to identify and mitigate security vulnerabilities.
* **Incident Response Plan:**
  + Developing and maintaining an incident response plan that outlines procedures for detecting, reporting, and mitigating security incidents.
* **Maintaining audit logs**:
  + Maintaining comprehensive audit logs of all security-relevant events, including user logins, data access, and configuration changes.
* **Implementing redundancy, failover mechanisms, and disaster recovery plans:** 
  + Which ensures high availability of the system and minimizes downtime due to security incidents or failures.
* **Configuration management practices:**
  + Implement secure configuration management practices to ensure that the system and its components are configured securely, reducing the attack surface.

# Management Plan //Xiankun Niu

## Objectives and Priorities

(Please describe your project objectives with highest priority first. Project Goals can include but not limited to complete all proposed (essential) features, deploy the software successfully, the software has no known bugs, maintain high quality, etc )

Setting goals in project management is a great way to improve the quality of the project. It is, by extension, also an excellent way to improve ourselves. For example, planning out the key objectives of the project is a great way to stay motivated, measure our growth, and prepare for the future.

In brief, project management objectives are the successful development of the project's procedures of initiation, planning, execution, regulation and closure as well as the guidance of the project team's operations towards achieving all the agreed upon goals within the set scope, time, quality and budget standards.

In order to make great project goals, we should follow the principles of **SMART** goals, which is defined as being **S**pecific, **M**easurable, **A**ssignable, **R**elevant, and **T**ime-based.

Here is the Objectives and Priorities of our project:

### Deliver a Functional Software Product (Highest Priority)

* **Objective**: Develop and deliver a fully functional software product that meets all specified requirements and user expectations.
* **Priority**: This is the highest priority as the successful delivery of a functional software product is the primary goal of the project.

### On-Time Project Delivery

* **Objective**: Complete the project within the established timeline and adhere to the project schedule.
* **Priority**: Timely delivery is crucial to meet business needs and maintain stakeholder trust. Any delays could have adverse consequences.

### Improve team communication and collaboration

* **Objective**: There is always room for improvement in team collaboration and communication, which makes this project management goal a great one to strive for.
* **Priority**: This is the basic and important priority as it can help team members adapt to each other's working style and pace, improving project development efficiency.

### Improve Productivity

* **Objective**: consider using the project management software which can offer simplicity and reliability and allow your team’s productivity to thrive. Objectives for this goal may include researching different vendors, signing up for product demos, and testing the various solutions on the market. ( For our project, we will use the **Monday** tool.)
* **Priority**: This is the top priority since with its project management features and functionalities, such tools help project managers manage multiple complex projects with ease.

### Budget Adherence

* **Objective**: Manage project costs effectively to stay within the approved budget.
* **Priority**: Staying within budget is a critical success factor and demonstrates financial responsibility.

### Quality Assurance

* **Objective**: Ensure that the software product is of high quality, adheres to industry standards, and is free from known bugs.
* **Priority**: Quality is paramount to meet user expectations, prevent post-launch issues, and maintain the software's reputation.

### Manage Stakeholders’ Satisfaction

* **Objective**: Continuously engage with stakeholders, gather feedback, and make necessary adjustments to meet their needs and expectations.
* **Priority**: Ensuring that stakeholders are satisfied with the final product is essential for long-term project success and future collaborations.

### Optimally Utilize Project’s Resources

* **Objective**: Everyone knows that projects are given limited resources to work with – budget, time, people, etc. The goal here is to find the best way to optimally utilize the resources that you’ve been able to secure for your project.
* **Priority**: This is a very high level priority since It is more than a project management goal, it is also a test of your skill to make the most of your available time, provided budget, and the member’s work.

### Scalability and Future-Proofing

* **Objective**: Design the software architecture with scalability in mind to accommodate future enhancements and updates.
* **Priority**: Its priority is not very high,but preparing for the future ensures that the software can evolve to meet changing business needs without major overhauls.

### Documentation and Knowledge Transfer

* **Objective**: Create comprehensive documentation for the software and ensure knowledge transfer to the support and maintenance teams.
* **Priority**: Documentation and knowledge transfer are essential for long-term software sustainability.

## Risk Management //Chandrasiri

Identifying and managing risks within a group is essential for achieving successful outcomes. Therefore we identified and listed below some main risks that need to be considered with our project.

Issues that are listed below can potentially cause problems on our project such as a delay of the schedule. Therefore we want to mitigate the identified risks as described below as soon as possible rather than waiting for them to confront us in the process of building our web application.

* **Fractious changes or uncontrolled additions to project requirements.**

To manage this risk,

* + establish a well-defined scope from the outset,
  + implement a formal change control process, and
  + involve stakeholders in decision-making.
* **Skill gaps**
  + Online training materials and
  + pair programming strategies are used to overcome this.
* **Ineffective communication among team members.**

In order to mitigate this issue, we foster

* + clear and open communication through regular meetings,
  + status reports, and
  + use communication technologies such as instant messaging, and email.
* **Personal issues**

Risks associated with personal issues and how to plan to manage them.

* + **Members leave**:
    - Invites for new members to join or
    - Appropriately reduce the application's functions
  + **Add new members:**
    - Implement a comprehensive onboarding process that includes an introduction to the project's goals, architecture, codebase, and existing documentation.
    - Provide access to relevant training materials.
    - Pair new members with experienced team members for mentorship and pair programming sessions.
  + **Members' weekly tasks are not completed**:
    - Ensure that team members are cross-trained to some extent.
    - Other members will temporarily take over their tasks and report it.
    - Refer to Risk(a) when members fail to complete tasks multiple times.
  + **A team member may fall sick**:
    - Cross-trained to some extent, so they can perform each other's tasks in case of absence.
    - Set up a backup person for all developments that fall on critical paths. And want to ensure that team members are
    - Go through the status reports of the module and identify the level of risk of delaying the module.
    - According to the information an appropriate member will temporarily take over his/her tasks or let it be done later by himself or herself.
* **Management issues**

We are ready to mitigate these challenges by conducting thorough;

* + Technical assessments,
  + Prototyping solutions, and
  + Allocating contingency time in project schedules.
* **Technical issues, such as scalability problems or integration difficulties.**

We are ready to take over this risk by;

* + Implementing robust testing and quality assurance processes, including manual testing, automated testing and code reviews.
* **Poor software quality**
  + Identifying strengths and weaknesses of each member continuously
  + Open discussions on taking responsibilities
  + Establishing as well as obeying code of ethics and standards for the team
* **Inadequate documentation and testing**

In order to mitigate this risk we are planning to ensure comprehensive documentation throughout the development process, including

* + Code comments,
  + User manuals, and
  + Architectural diagrams.

**Risk Management Sheet Link:** [**Risk Management Sheet (CS673\_SPPP)\_Team3**](https://docs.google.com/spreadsheets/d/1vNNvFCzozsFB329RsX6QYxaJxrvODl-9SQ41gdoASFI/edit?usp=sharing) //Chandrasiri

## Timeline (this section should be filled in iteration 0 and updated at the end of each later iteration)//Siyuan Xue

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| 0 | Essential:  1 Customer needs analysis  2 Restaurant demand analysis | 1. Overall content of the design project  2. Unified team member environment configuration | 20 |
| 1 | Essential：  Home Page: Users should be able to see the home page and be able to navigate to other functional pages.  Menu: The user can view different dishes and select the desired one.  Shopping Cart: The items selected by the user should be added to the shopping cart, allowing the user to modify the quantity or remove items.  Confirmation Page: Users should be able to review and confirm their order details before submitting their order.  Login Page: Registered users should be able to log in to the system.  Sign Up Page: New users should be able to register for an account.  Email Verification for Registration: Users should receive a verification email when registering to confirm their email address.  Optional：  Consider adding user comments or evaluation functions.  Provide recommendations or special dish displays.  Disable：  Provide users with profile editing capabilities. | User Authentication  Implement the login and registration process and ensure that passwords are stored encrypted.  Design the email verification process, including sending verification emails, verification links and verification feedback.    Database Design  Design and create a database model including user information, menu items, shopping cart details, etc.  Implement order history recording and viewing functions.    Shopping Cart Logic  Allows users to add, remove or modify the quantity of items in the shopping cart.  Implement the automatic calculation function of the total price of the shopping cart.    Order Confirmation Logic  Provide a clear order overview including selected items, quantity and total price.  Allow the user to confirm the order and go to the payment page or return to the cart to make changes.    Integration  Ensure smooth transitions between homepage, menus and shopping cart.  Implement status checks for login and registration pages, for example, if the user is logged in, jump directly to the home page. | 80 |
| 2 |  |  |  |
| 3 |  |  |  |

# 

# Configuration Management Plan

## Tools

For our project, we will use Jetbrains Space and GitHub as the version control tools. This will allow us to collaborate efficiently and track changes in our code. We will all use JetBrains’ PyCharm as our IDE since it provides a simple GUI with built-in plugins for database and web pre-view.

For the CI/CD tools, we will be using TeamCity by JetBrains, which is a powerful and user-friendly tool that can be integrated with a remote server. This will enable us to automate our testing and deployment processes and ensure quality and reliability. For the container tools, we will use a normal Python virtual environment, which is sufficient for our needs and does not require additional complexity. For the SAST or DAST tools, we are still evaluating the options and will decide if the process need one.

* 1. Code Commit Guideline and Git Branching Strategy

Trunk-based development is our preferred approach for code commit and branching. It means that we all work on the same branch, the trunk, and integrate our code changes into the main codebase as soon as possible. This way, we avoid the hassle of managing multiple branches and we can deploy our code faster and more often, which is preferred in this situation.

## Deployment Plan if applicable

We have decided to use the following services and platforms for our web app development and deployment:

- For domain registration, we will use Namecheap.com, which offers affordable and reliable domains. The estimated cost is 10$ per year.

- For SSL and DNS, we will use Cloudflare, which provides free and secure certificates and fast DNS resolution.

- For cloud server, we will use Azure for students, which gives us access to a range of cloud services and resources for free or at a low cost due to its connection with school.

- For the database, we will use SQLite, which is a lightweight and embedded SQL database engine that does not require a separate server process.

# Quality Assurance Plan //Jialin Li

## Metrics

(Describe the metrics to be used in the project to measure the quality of your software. Each metric should be measurable and quantifiable. Examples of metrics include product complexity (LOC, # of files, # of classes, # methods, cyclomatic complexity, etc.) , defect rate (# of defect per KLOC), # of test cases, test case pass rate, cost (# of person hours used), # of user stories completed, etc. **The result of these metrics should be reported in the progress report/ iteration summary sheet.**)

| Metric Name | Description |
| --- | --- |
| Defect Rate | Number of defects per thousand lines of code (KLOC).  Ensure that the defect rate is below the predetermined standard. |
| Number of Test Cases | The number of test cases executed in the project.  Make sure all features are fully tested. |
| Test Case Pass Rate | Percentage of all test cases that pass.  Maintain a high pass rate. |
| Code Complexity | Measure code Complexity using methods such as Cyclomatic Complexity.  Keep code complexity manageable. |
| Cost | The number of human hours used by the project.  Ensure that the cost does not exceed the project budget. |

* 1. Coding Standard

1. Use lowercase letters for variables, separated by underscores (e.g. total\_orders).

2. Use the hump naming method for class names (for example, OrderSystem).

3. Each file should contain a comment block in the header of the file, which should include the file name, author, creation date, and description of the file.

4. Use multiline docstrings to describe the functionality of modules, classes, and functions.

5. Try to avoid using excessively long functions and methods, and consider splitting complex functions and methods into multiple simple functions and methods.

6. Use unit tests to verify the correctness of your code.Each module and method should have a corresponding test case.

## Code Review Process

(Everyone should review all documents to be submitted. Here you will mainly describe how the code review will be done. Who will review the code, e.g. design or implementation leader will review all code or team members review each other’s code. Do you use pull requests for the code review? Is there a checklist to help review? What feedback should the reviewer provide?)

1. Submit the Code Procedure:

Developers submit their completed work to a feature branch in the version control system (e.g., Git).

Standardization: Ensure that the submitted code aligns with the task requirements.

2. Create a Pull Request (PR) Initiation:

Developers create a PR to merge the feature branch into the main/master branch.

Content: The PR must include a comprehensive overview and the purpose of the changes for clarity and traceability.

3. Code Review Reviewer Assignment:

Design or implementation leaders, or assigned senior team members, will conduct code reviews.

Review Areas:

Adherence to Coding Standards: Check if the code complies with the project's predefined coding standards and guidelines.

Relevance and Necessity: Verify that all changes and additions are essential for the feature’s functionality.

Clarity and Quality: Assess if the code is clear, maintainable, and free of errors.

Test Passing: Ensure all functional and unit tests pass successfully. Documentation and Comments: Evaluate the completeness and clarity of inline comments and external documentation.

4. Provide Feedback Feedback Nature:

Constructive, specific, and aimed at helping the developer enhance their code quality.

Communication: Feedback should be communicated clearly within the pull request comments.

5. Revise and Merge Code Revision:

Developers revise the code as per the feedback received. Final Review and Merge: Once the revised code meets all quality standards and passes all tests, it's approved for merging into the main branch.

6. Close Requests and Record Important Information Documentation:

Log any crucial information, decisions, or follow-up actions arising from the review.

Closure: Close the pull request post-merge, ensuring all discussions and actions are resolved.

## Testing

(Both manual testing and automated testing should be considered. Both unit testing and integration testing should be considered. Briefly describe the testing tools/framework to be used, the personnel involved (e.g. the QA leader will focus on the integration testing and each developer will unit test their own code), when and what types of testing will be performed, the testing objectives, etc)

**1. Test strategy**

a) Unit testing

Objective: Test individual components or modules to ensure they function as expected in isolation. Methodology: Developers will write unit tests for their own code. Tools/Frameworks: Utilize Python's unittest framework or pytest for more sophisticated test requirements.

Responsibility: Each developer is responsible for their unit tests.

b) Integration testing

Objective: Assess the interaction between different modules or components and ensure they work together as intended. Methodology: Focus on interfaces and data flow between modules.

Tools/Frameworks: Use integration testing tools that can handle multiple components, potentially pytest along with specific plugins or custom scripts. Responsibility: The QA leader and a team of testers will oversee integration testing.

c) System test

Objective: Validate the complete and integrated software product. Methodology: Use test cases that simulate real-world usage to assess functionality, performance, and security. Tools/Frameworks: Tools like Selenium (for web applications), Load Testing tools like JMeter, and Security Testing tools like OWASP ZAP.

**2. Test environment**

Setup: Mirror the production environment to ensure realistic testing conditions. Components: Include all necessary hardware, software, network configurations, and databases.

**3. Test data**

Preparation: Create datasets that cover normal, boundary, and erroneous conditions. Variety: Ensure data is diverse and voluminous to test different scenarios effectively.

**4. Test tools**

Selection: Choose tools based on project requirements, ease of integration, and team familiarity (e.g., pytest, Selenium).

Automation: Implement CI/CD pipelines using tools like Jenkins or GitHub Actions to automate testing and ensure continuous quality control.

**5. Test plan**

Details: Outline specific tests, objectives, responsibilities, and timelines.

Review: The plan should be reviewed and approved by project stakeholders.

Update: The test plan is a living document and should be updated as the project evolves.

**6. Test execution**

Process: Execute tests as per the test plan.

Documentation: Document all test cases, results, and any deviations.

Defect Tracking: Log and track all defects in a defect tracking tool.

**7.Test report**

Testing is a critical part of software development that ensures quality and reliability.

A structured approach to testing with clearly defined responsibilities and tools is essential for successful project delivery.

Continuous feedback and iterative improvement of the test strategy based on test outcomes should be incorporated.

## Defect Management

(Describe the tool to be used to manage the defect (e.g github issues). The types of defects to look at. The actions or personnel for defect management. )

1. Defect identification

Defects can be discovered at any stage of the project. All team members should be encouraged to report defects.

2. Defect record

Each defect found should be recorded in the defect tracking system.

The defect record should include detailed information about the defect, such as defect description, degree of impact, etc.

3. Defect classification

Defects should be classified according to their severity and priority.

Severity: Determines the impact of the defect on the system.

Priority: Determines how urgent it is to resolve the defect.

4.Defect investigation

The leader should investigate the defect, determine its root cause, and actively collaborate with other team members to find a solution.

5.Defect repair and validation

Once a solution has been identified, defects should be fixed immediately. Fixes should be made on a separate branch and passed code review and testing.

6.Defect reporting and analysis

Periodically generate defect reports that summarize information about defects found and resolved.

Perform defect analysis to identify defect patterns and root causes.

# 

# References

[www.github.com](https://www.github.com/) - Tutorials for quality assurance programs can be found on GitHub.

# Glossary

DNS: Domain Name System. A system that translates human-readable domain names (such as www.example.com) into numerical IP addresses (such as 192.168.0.1) that computers can understand.

SQL: Structured Query Language. A language that allows users to manipulate and query data stored in relational databases.

SSL: Secure Sockets Layer. A protocol that encrypts the communication between a web browser and a web server, ensuring the privacy and integrity of the data.