Software Quality Deliverable 2.3

Parking Garage

_

Fixed/Improved Analysis Design

[we used the same design!]

_

Quality Report

Product Quality Metrics

Product Operation

Quality: Correctness

Goal: The number of correctness issues should be less than 5.

Metrics: The functionality issue reports should less than 5 in the whole lifecycle.

Validation Results: I ran the system on my own server which is on my computer for whole day. There is no failure report recorded in the log files, but there are some exceptions when the system was running. The most exceptions relate to the operation of query information from the database. Some of exception was caused by null point exception, because there was no related information recorded in the database. These kinds of exception will be caught by the program, so the error message will not be showed to users, which means there is no negative effect for users' operation. The other exception was about the failed connection to the database. Our database server is separate from the code server, so there is a problem of network connection failure when the code server connected to the database server. For these kind of exception, the system will showed failure page to the users in order to tell them the problem of connection, and they can try again late.

Quality: Reliability

Goal: The system shall work 99% accurately

Metric: The crashing frequency should be less than 3 per year

Validation Results: Pylot which is a stress test tool can set the number of agents and the duration. We set the 400 agents request the service during 10 seconds, the result shows as following:

Pylot - Performance Results

report generated: 04/05/2020 22:26:58
test start: 04/05/2020 22:26:47
test finish: 04/05/2020 22:26:52

Workload Model

test duration (secs)	5
agents	400
rampup (secs)	0
interval (millisecs)	0

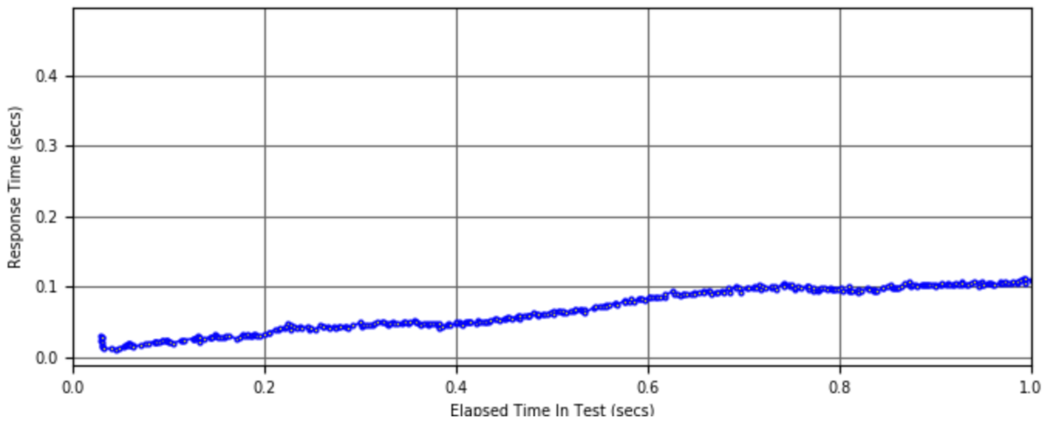
Results Summary

requests	1589
errors	0
data received (bytes)	8868209

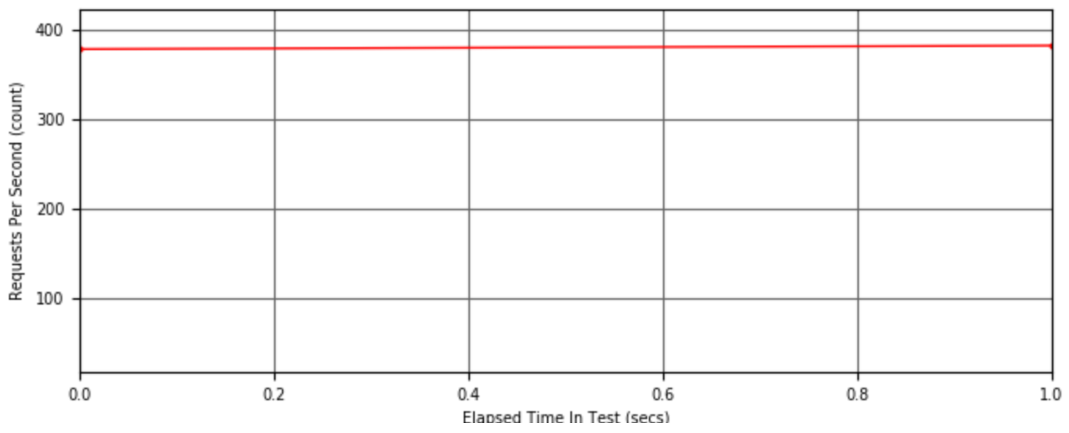
Response Time (secs)		Throughput (req/sec)	
avg	0.244	avg	317.800
stdev	0.132	stdev	157.842
min	0.011	min	36
50th %	0.232	50th %	382
80th %	0.376	80th %	404
90th %	0.420	90th %	404
95th %	0.444	95th %	404
99th %	0.464	99th %	404
max	0.473	max	404

The average response time is 0.244 secs and there is no errors in this process. Therefore, we believe this system is reliable enough. Here is the response time graph and throughput graph:

Response Time



Throughput



Quality: Learnability
Goal: The system should be easy to learn and to use.
Metric: A new user should be able to use the system less than a day's training.
Validation results: new users of the system learned in approximately half hours how to use all of the functionality of the prototype: namely, registering, logining in, and making a reservation.

Quality: Integrity

Goal: They data should be integrated for more than 3 years

Metric: The data leakage accidents should be less than 1 in 3 years

Validation results: There has not been full audit done. The connection between the web server and the database management system is based on TLS 1.3, which strongly encrypts the data between the web server and the database management system. However, the password for logging in the database system is short and easy-to-guess. Moreover, the web application in our system is based on HTTP protocol directly without any encryption. All the packets sent between the clients and the server can be easily captured and parsed. The system has no SQL injection checking to the content of submitted forms. Thus, An SQL injection attacking is possible to our system.

Quality: Usability
Goal: The system should be user friendly.
Metric: There should be no more than three complaints from users when trying to learn the system.
Validation results: Users were dissatisfied with the lack of detail in register. When you fail to register, the system does not notice any reason. Work should be scheduled to add the necessary details.

Quality: Efficiency
Goal: The system should quickly solve the intended problems.
Metric: The system should finish all data query and manipulation in less than one second.
Validation results: Most responses were returned from the server on the order of seconds. There are certain operations in the Database Accessor that could be done asynchronously where they are currently synchronous which could improve end to end latencies.

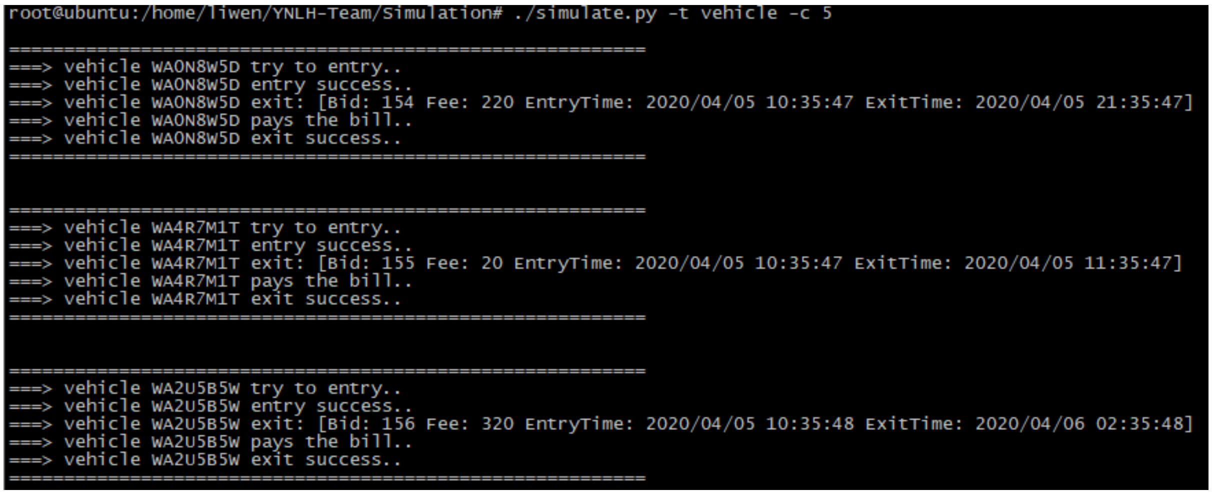
Product Revision

Quality: Maintainability

Goal: The system should be easy to maintain and it should be ensured changes would not lead to defects.

Metric:
<1> The program should not contain lexical and design anti-patterns[3][6], and it should be well modularized, and follow the principle of high cohesion and low coupling.
<2> the program should contains error correction mechanism, such as automated test unit to ensure each new change would not break down previous functions.
<3> the program should be implemented with low cyclomatic complexity.

Validation results:
<1> we utilize Spring-MVC as our base model, the whole design is layered, it is easy to integrated new features.
<2> we utilize Junit as our unit test tool to ensure correction of functions, which also helps avoid defects brought by changes during maintaining.
<3> we utilize automated test script (python) to construct error correction mechanism, which helps avoid defects at system (requirement) level. (an example showed as below)



```
root@ubuntu:/home/liwen/YNLH-Team/Simulation# ./simulate.py -t vehicle -c 5

=========================================================
===> vehicle WAON8W5D try to entry..
===> vehicle WAON8W5D entry success..
===> vehicle WAON8W5D exit: [Bid: 154 Fee: 220 EntryTime: 2020/04/05 10:35:47 ExitTime: 2020/04/05 21:35:47]
===> vehicle WAON8W5D pays the bill..
===> vehicle WAON8W5D exit success..
=========================================================


=========================================================
===> vehicle WA4R7M1T try to entry..
===> vehicle WA4R7M1T entry success..
===> vehicle WA4R7M1T exit: [Bid: 155 Fee: 20 EntryTime: 2020/04/05 10:35:47 ExitTime: 2020/04/05 11:35:47]
===> vehicle WA4R7M1T pays the bill..
===> vehicle WA4R7M1T exit success..
=========================================================


=========================================================
===> vehicle WA2U5B5W try to entry..
===> vehicle WA2U5B5W entry success..
===> vehicle WA2U5B5W exit: [Bid: 156 Fee: 320 EntryTime: 2020/04/05 10:35:48 ExitTime: 2020/04/06 02:35:48]
===> vehicle WA2U5B5W pays the bill..
===> vehicle WA2U5B5W exit success..
=========================================================
```

Quality: Flexibility

Goal: The system can be easily changed
Metric:

The changing in some parts should not affect the whole running.
Validation results:
In this system, it utilizes Spring-MVC model. Under this model, Each team member is assigned to develop a separate module. There are two main sections, front-end development and back-end development. For the front-end developers, they just need to focus on the page design and decide the same message format with the back-end developers. For the back-end developers, they are assigned different development of module. They will create their own code file to develop. Thus, each changing in team mates' own part will not affect others' functionality. Besides, Spring-MVC model is very friendly for developer. Developer can use extend library just depends on configuration and operate database by configuring xml files.

Quality: Testability

Goal: The system should support automated test and be easily testable.

Metric:
<1> Test cases should be complete, including functional cases and non-functional cases.
<2> Feature correction metric: each feature (requirement) should be tested through manual operation and automated test script, including normal/abnormal cases.
<3> Non-feature correction metric:
The program should provide performance test script to verify the throughput.
The program should provide stress testing script to test system stability.
Validation results:
<1> we integrate manual test operations in an testing-purpose web page, which is easy to use.
<2> we provide python scripts to test all function/non-function features.

Product Transition

Quality: Reusability

Goal: Parts of the system should be able to be reused in another system
Metric:
Modules in the system can be used in another system in less than 1 week configuring
Validation results:
The system is managed by Maven which is a useful project management tool. It can package the source code as a war file. Therefore, it can be add to other system as a extend library. Each source file can be reused in another system.

Quality: Portability

Goal: The system should be easily moved to a new platform

Metric: The system should work correctly on different devices, operating systems, database systems.

Validation results: This is a web-based application. All devices and operating systems which have a web browser can access the system without any issues. All queries and operations to the database system are based on standard SQL statements. There is no issue for different relational database systems. However, the system cannot moved to non-relational database systems without modifying the database query and operation statements.

Quality: Interoperability

Goal: The system should be able to interaction with other systems
Metric:
The system should be able to work with other data management systems
Validation results:
Each interface of the system is designed as the standard interface. Therefore, any system can interaction with the system by connecting to the standard interface. During the test of analog sensor interface, we try to use python script to connect the system, the system can effectively carry out feedback.

Quality Goals

See the table of quality goals below.

Product Quality	Quality Goals	Quality Metrics	Strategy
Product Operation			
Correctness	The number of correctness issues should be less than 5	The functionality issue reports should less than 5 in the whole lifecycle	Provide beta version to allow customers to experience and provide feedback
Reliability	The system shall work 99% accurately	The crashing frequency should be less than 3 per year	Stress testing
Learnability	The system should be easy to learn and to use.	A new user should be able to use the system less than a day's training	See the detailed description above.
Integrity	They data should be integrated for more than 3 years	The data leakage accidents should be less than 1 in 3 years	Unit test on normal cases and boundary cases
Efficiency	The system should quickly solve the intended problems.	The system should return search results in less than one second	See the detailed description above.
Usability	The system should be user friendly.	The system should have well designed interface.	See the detailed description above.
Product Revision			
Maintainability	The system should be easy to maintain and it should be ensured changes would not lead to defects.	*See the detailed description above*	Code Review, Static code analysis tools Error correction tools
Flexibility	The system can be easily changed	The changing in some parts should not affect the whole running.	Utilize Spring-MVC model
Testability	The system should support automated test and be easily testable.	*See the detailed description above*	Mutation and Unit testing tools Automated test scripts
Product Transition			
Reusability	Parts of the system should be able to be reused in another system	Modules in the system can be used in another system in less than 1 week configuring	Standard interface design and Objected Oriented Design
Portability	The system should be easily moved to a new platform	The system should work correctly on different devices, operating	Standard interface design

		systems, database systems	
Interoperability	The system should be able to interaction with other systems	The system should be able to work with other data management systems	Standard interface design