Liu Hoi Pan 1155127464 CSCI3100 Assignment 1

Q1

(a)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Quality | Internal | External | Product | Process |
| Correctness | √ | √ | √ | √ |
| Reliability | √ | √ | √ | √ |
| Robustness | √ | √ | √ | √ |
| User Friendliness | X | √ | √ | X |
| Verifiability | √ | √ | √ | √ |
| Maintainability | √ | √ | √ | √ |
| Repairability | √ | X | X | √ |
| Evolvability | X | √ | √ | √ |
| Reusability | √ | X | X | √ |
| Portability | X | √ | X | √ |
| Understandability | √ | √ | √ | √ |
| Interoperability | X | √ | √ | X |
| Productivity | √ | X | X | √ |
| Timeliness | √ | √ | X | √ |
| Visibility | √ | X | X | √ |

(b)

First three aspects are dependability.

Correctness means the program is functionally correct and it is consistent with its specification. If the program does not meet the specification that users want, or the user’s requirement or product requirement, of course it affects external and product quality. Moreover, developers need more time to debug and correct the program by various techniques like testing, so it affects internal and process quality.

Reliability means the probability that software will operate as expected. If a software is correct, then it is reliable. If the program is reliable, it means it seldom contains bugs. Users are satisfied. Also, developer need to work on maintenance more carefully.

Robustness means the program behaves reasonably. If the program doesn’t work, user never satisfy about it. Developers need to fix it, so robustness affects all four qualities.

A software system is user friendly if users find it easy to use, so it affects external quality and product quality. However, the coding inside the system is not related to user friendliness.

A software is verifiable if its properties can be verified easily. Usually, it is not related to external and product qualities if users don’t care about it. But sometimes, the verifying security of applications may be required by users, so it is related. Moreover, verifiability is related to internal and process qualities. It is because if a software can be easily verified, it means the coding can be easily checked and changed, this makes developers more convenient. Also, the test data can be easily checked which increases the process qualities.

Maintainability includes repairability and evolvability. A software is maintainable if it is easy to modify. Thus, a software with high maintainability is more convenient for developers. The internal and process qualities increase. Moreover, maintainability includes perfective maintenance which also affect external and product qualities. As the software becomes more perfect, the users become more satisfactory. Therefore, all four qualities are related to maintainability.

Repairability and evolvability can be considered as a part of maintainability. A software system is repairable if its defects can be corrected easily. This is convenient for developers, so it is related to internal and process qualities. Evolvability is both a product and process quality. It is because the software evolves to be a better product, satisfying users’ requirement, as well as the software is easier for developers to process the coding.

Reusability is like evolvability. In product reuse, we use a product with minor changes to build another product. It is more convenient for developers. It is because if the program is reusable, developers can easily copy and manage it. Therefore, internal and process qualities increase.

Software is portable if it can run in different environments. Portability is economically important because it helps different software system across different environments to attracting investments. This is related to external quality. Also, portability can be considered as a special case of reusability. Developers may find the software easier to process.

Understandability means the program is easy to understand or not. A program can be easily understanded and modified means the program has high internal and product qualities for debugging and maintenance purpose, as well as evolvability and verifiability of the software. Moreover, understandability can be extended to external and product qualities because it may be related to user friendliness. If a program is more understandable for users, users can use it more easily.

Interoperability means the ability of a system to cooperate with other systems. It is related to external and product qualities because users may use the system with other systems together, overcoming the drawbacks of just using one system. It is not really related to developers’ aspects.

Productivity is a quality of software production process. It is quite different to other qualities measuring the products. Productivity measures the efficiency and performance of a software process. Users won’t care about the software process when they are using the software, so it is not related to external and product qualities. Besides, having higher productivity means there may be a larger team of developers, which is quite useful in internal and process aspects. It is because developers have fewer workloads to manage the code.

Timeliness is a process-related quality. It measures the ability to deliver a product on time which is quite similar to productivity. Therefore, timeliness is related to internal and process qualities. In addition, if a product is delayed delivering, customers may feel disappointed about it, decreasing the external quality. The product quality is still the same. Delaying the deliver of product won’t affect its function.

Visibility means the steps of development process and its status can be easily viewed. It is related to developers since they can easily know which part of the code requires managing, so it is related to internal and process qualities. Users won’t care about the how the software is developed, so it is not related to internal and product qualities.

Q2

(a) T

High maintainability means the code is easy to modify to fix bugs and add new features. They are just the definition of repairability and evolvability. The backward is still the same. Also, high repairability means the code is easy to modify to fix bugs. If it is the case, the code is easy to manage, it should be easy to modify to add new features too.

(b) F

If the specification is not good, meeting the specification doesn’t mean the program is robust. The program may behave unreasonably.

(c) T

They are qualities related to time. If a developer team has high productivity, their efficiency is high. It means they can create many products in a certain time. Their timeliness is also high. They deliver the product on time.

(d) F

Reliability is like correctness, so similar to question 2(b), we have false.

(e) T

If the performance of a software is low, it means the software system uses too much memory, it may be too expensive to run. Users may find the software doesn’t meet their needs, or they hardly use the software.

(f) F

The program is understandable just means developers can easily modify the code. It is not related to whether the code can operate in different operating system.

(g) F

The software is verifiable means it can be easily tested. It is not related to whether the program meets the software specification.

(h) T

The software making process is visible means the processing status is known by developers. Developers can know status of the process every time require, at least, high understandability coding.

(i) F

A software can be used may not be reused. For example, a software can be used since it has good performance. But the software can be difficultly to modify to a new software.

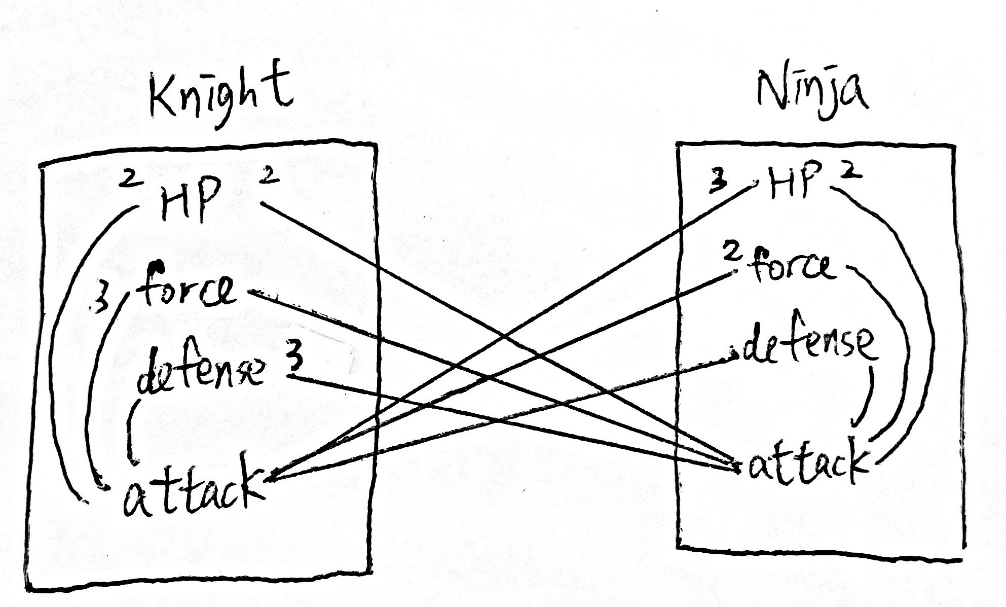
(j) T

Portability is a special case of reusability. If the software can run in different environments, it can be easily modified too.

Q3

(a)

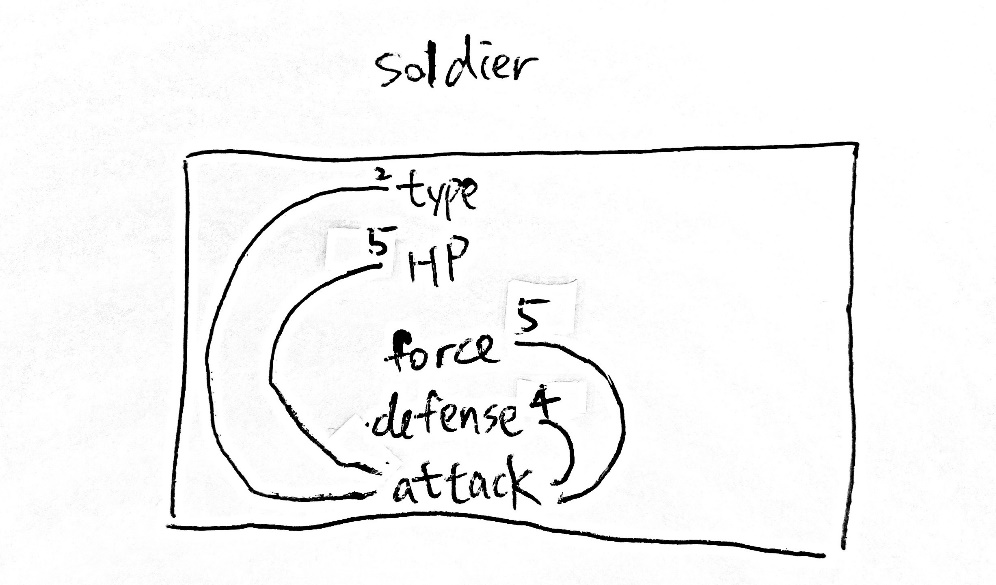
Tony’s code



Cohesion: 10

Coupling: 12

Mary’s code



Cohesion: 16

Coupling: 0

Mary’s code is better. The key to design modularity is to achieve high cohesion and low coupling. Highly cohesive module makes components in a module closely related to another. Module becomes more meaningful. Low coupling makes each module can be more understandable separately.

(b)

If we do not inherit certain soldiers from the parent class, it means we implement Tony’s code but not Mary’s code.

First, in the aspect of modularity, we will have lower cohesion and higher coupling. It is because we will create a new class, *Dragon*. This new class will have a function, *attack*, which links to *knight* and *ninja*. The coupling lines will become more and more. At the same time, there exists some cohesion inside *Dragon*, for example, between *attack* and *HP*, between *attack* and *force*, etc. But coupling lines are more. It makes the program look more complicated, not satisfying the keys of modularity: high cohesion and low coupling.

Second, in the aspect of anticipation of change, it is difficult for developers to evolve the code. If we create a new class, *Dragon*, we need to add different functions in every class to link to *Dragon*. For example, *attack* of *Knight* to *Dragon*, *attack* of *Ninja* to *Dragon*. It causes huge workload. Imagine if we need n more classes, then we need to add n + (n-1) + (n-2) + … + 1, which is O(n2) process (*attack* from first two functions to third, *attack* from first three functions to fourth, …, *attack* from first n-1 functions to n). This is of course a huge process. But if we implement the code with Mary’s version, we just need to add n conditional functions with different argument (int *type*). It is O(n) process. As we can see from here, the evolvability, maintainability and reusability are well improved. Hence, we can see a huge workload problem if we don’t inherit certain soldiers from the parent class.Q4

(a)

Class Game {

Gameboard[];

timestamp;

souls;

gameStatus; //if someone win, change it

}

Class headquarter {

status; //check if there us any soldier in the headquarter to produce soldiers

generateSouls();

soldierProducing();

gameOver()

}

Class castle {

castleSouls;

generateSouls();

getSouls(); //if there is soldier on the castle, gain its soul

}

Class Soldier { //base on Mary’s code

type; //Ninja, Knight, Demon are included here

HP;

force;

defense;

soldierAttack();

soldierMoving();

soldierDie();

}

(b)

I will use extreme programming.

First, using extreme programming is simple. Extreme programming improves software quality and response customer requirements. Once customers require the implementation of new features, I modify the code immediately. In this case, if I have to implement new soldiers, I just need to add more type, as well as add new conditional statement. Then, I can quickly release the new product and get new response from customers. This programming model greatly improve the efficiency and productivity of the project.

Second, using extreme programming require low cost. Especially the time cost is low. It is because I don’t need to write a detailed specification to customers to indicate the function of each part in my program. This saves a lot of time.