Mirza Ovcina 200272124
CS373 Assignment #2

1.
Separating software into 3-later architecture style has following benefits:

Maintainability and upgradability: individual software layers can be altered without affecting different parts of the program. For example, interface layer can be completely altered with minimal effects on logic and data tiers.

Increases portability: since layers are independent of one another, if there are hardware or operating system incompatibilities, only layers that are affected need to be altered. Consider a website built using (presentation) HTML, (logic) PHP, (data) MySQL. If the website is to be migrated to a server, which lacks PHP interpreter support or has an incompatible version of PHP, the developer need only change logic layer.

Parallel development: multiple different development teams can work on the project concurrently.

Scalability: because layers are independent, developers and system administrators are able to add or remove resources depending on server traffic.

2.
Low coupling:

Low coupled or loosely coupled software is built in such a way that its modules/components have minimal dependencies on other modules.

Concrete example:
A non-software example of loose coupling could be an enterprise level server. This type of computer has redundant components that can be modified, upgraded or removed during runtime without affecting overall system stability. In contrast, a tablet computer is a tightly coupled system where all components are intercoupled.

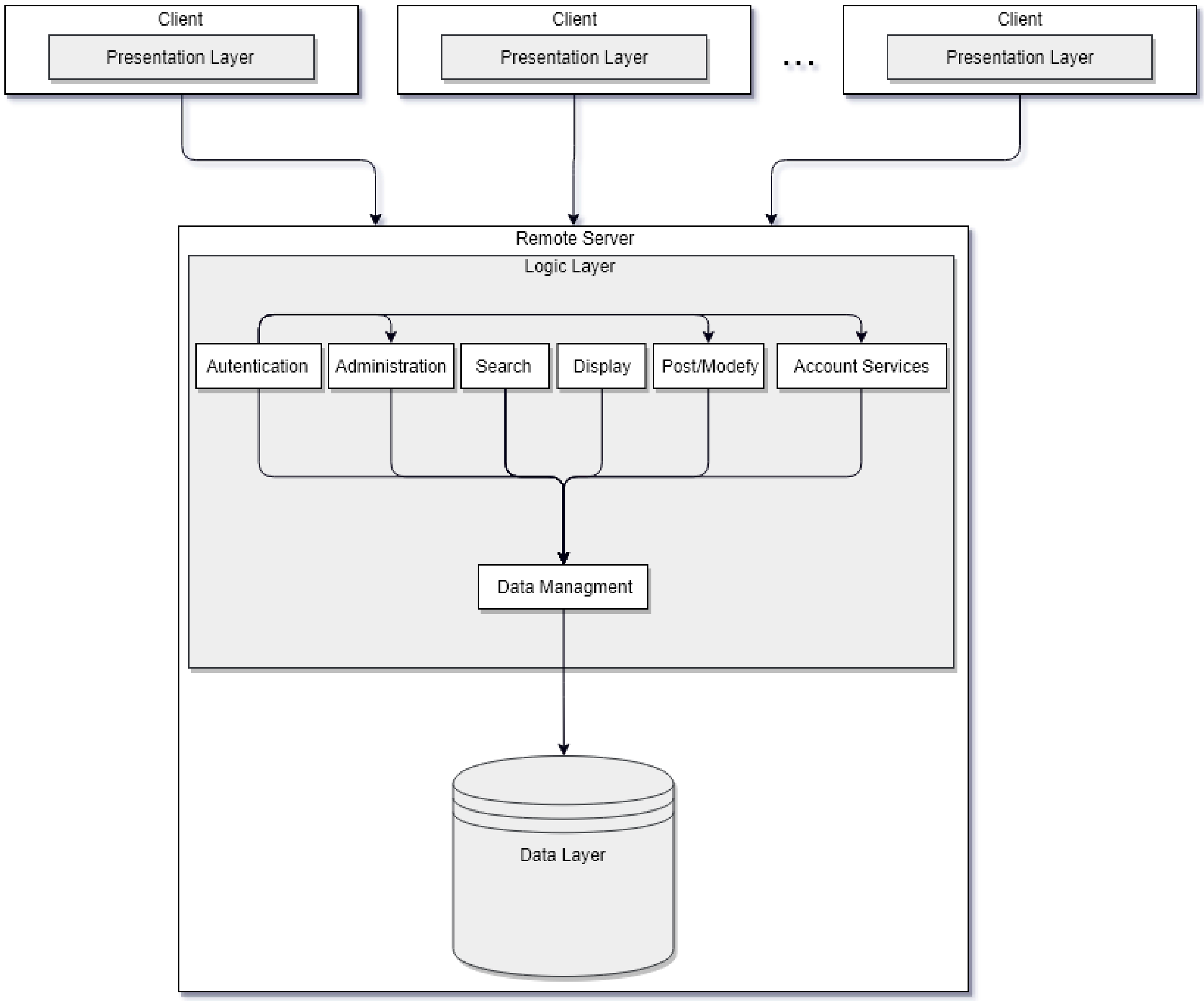
A software example, could be logic layer of a website which has classes for features like database manipulation and session functionality and a class that serves as an interface bonding all other classes together.

Cohesion is a measure of a bond and cooperation between modules in a software. For example, in object oriented paradigm, strong cohesion is an attribute that describes classes whose methods are specialized in solving specific problems and have high degree association amongst themselves.

Concrete example:
A non-software example, could a hospital that staffs many specialized surgeons. Each of the surgeons will perform surgeries in their respective fields and will perform no other task.

Software example, may be in the form of a GUI class which has methods are responsible for drawing menus, windows, prompts and has no offer no other functionality.

3.



4.
The goal of software testing is the following:

1. Discovery of errors/bugs which includes load time, runtime and logical errors.
2. Verifying that the requirements are satisfied and software solves the problem
3. Make sure all modules are functioning properly and communicate with each other

5.
Black box:

This type of testing assesses input and output of a system without regard for internal workings. Validity is assessed based on output given correct or incorrect input. Because the internal code is not examined, this type of test can be performed by programmers and non-programmers.

White box:

This test assesses and takes into account internal working and logic of the system. Tester inputs valid or invalid values into the system and traces the execution path thought the code and verifies the output. Testers performing white box testing need to have some familiarity of programming languages and are therefore usually the programmer themselves.

6.
Following are five types of software maintenance:

1. Corrective: Repair of error which are discovered after software is released
2. Adaptive: Modifying software to achieve compatibility with changing software and hardware environments
3. Evaluative: Addition or modification of features based on changing user requirements
4. Perfective: Performance or usability improvements though optimization.
5. Preventative: Revision or addition of documentation and code comments as well as increasing code modularity.

7.
Top down

This type of software design strategy emphasises understanding, decomposition and abstraction of problems. Planning is iterated in a stepwise fashion until a clear solution is found and implementation step can begin. One major befit of choosing this type of software development strategy is that most problems can be resolved very early in the development stage. However, if a serious issue is discovered during the implementation stage, extensive rework to the original plan may be needed. Another drawback to this design approach is a lengthy delay in release of early prototypes.

Bottom up

Bottom-up design is a polar opposite of top down. This design approach starts with the immediate implementation of simple systems and builds upon it adding complexity and functionality until the software is completed. This strategy benefits from having early testing prototypes and allows immediate testing and debugging. On the other hand, this approach can be disastrous to the overall project if there is a significant roadblock in the implementation the developer team is unable to overcome.

8.

Validation

Validation part of software testing validates weather the software produced is in-line with requirements, qualities and correctness which were drawn up during the planning stages.

Verification

Verification on the other hand, validates if the current state of software design is consistent with step that preceded it (ie. in waterfall design).

9.

Modified waterfall design embodies all qualities of traditional waterfall design with addition of backtracking, validation and verification. In contrast to traditional waterfall model, modified waterfall allows design team to “backtrack” or move back one or more steps in the design depending on the need.

10.

Evolutionarily prototyping

Prototype created and continuously updated with features and is presented to the customer for validation. Purpose of this type of prototype is to ensure the project is on the right track and the customer has opportunity to give feedback and clarify any complicated requirement details.

Throwaway prototyping

This type of prototype is developed to help developer team understand complex or convoluted requirements and gauge weather their understanding is aligned with requirements given by the customer. Once requirements are understood, throwaway prototype is discarded.

11.

This type of development model is used in extremely complex, expensive and mission critical software development projects. Model uses a four cycle development plan that incorporates risk analysis and prototyping at each step to ensure safety and reliability of the software produce. Early cycles are devoted to risk analysis and potential problem resolutions. These layers also produce prototypes and information that is used by layers following. Late cycles are consisting of more risk analysis and prototyping, and of course, software development.

12.

User (external)	***Developer(internal)***	***Management***
correctness	reusability	reusability
safety	performance	productivity
robustness	portability	visibility
performance	maintainability	timeliness
user friendliness		
security		

13.

Safety:

Safety quality generally pertains to critical software system and it appraises software's ability to prevent hazardous operations by enforcing safeguards or other means. It is imperative this type of software be free of errors, is easy to use, and provide clear and concise communication with the user.

Robustness:

This quality assesses software's ability to tolerate invalid user input without crashing and instead give user opportunity to enter valid input.

Security:

Software's ability to thwart unauthorized access to date both physically and virtually. This can be accomplished through the use of data encryption, firewalls, authentication and more.

Portability:

Measure of software's ability to run on different operating systems, hardware or combination of both.

Maintainability:

Maintainability assesses software's modularity, upgradability, repair and extendibility qualities. Software that uses architecture which is compliant with aforementioned qualities will have high degree of maintainability. Also OO software that employs inheritance and class abstraction will be maintainable.

14.

Because of the underlying JVM (Java Virtual Machine) computing environment, most software written in Java is portable across all popular operating systems including Windows, Mac OS and many distributions of Linux. Additionally, Java code can be uses in development of web applications and some of the code can be translated directly into JavaScript (jsweet) allowing almost anyone with a browser to run Java code.

15.

By using strongly vetted classes and functions that are reused from previous projects or from libraries such as C standard library, the overall reliability and corrected of current project increases. Some functions such as those found in cmaht.h have been used for years and have thousands of combined hours of testing backing their validity. Comparing to a similar function written by single programmer which may only have few test cases validating their operations, it is easy to see that quality reused software will be more economical, reliable and more importantly, correct when compared to writing code from scratch.

16.

Microsoft Visual Studio

As a software suite designed specifically for professional programmers and computer science students, this type of software has somewhat different quality requirements then mainstream computer programs.

Probably the most important quality for the developers and managers of Microsoft's Visual Studio is **functionality**. Being Microsoft's flagship code development (IDE) tool, Visual Studio is filled with diverse features and support for number of different programming languages like C/C++ ,functional language F#, Microsoft's own .NET languages and many more. All of programming languages are complimented by Intellisense, a real time code analysis and completion tool. Furthermore, Visual Studio also has extensive debugging and profiling facilities.

Following functionality, **robustness** is most likely next on the must-have quality list.

Due to the business environment Visual Studio is used in and the significance of the projects built using it, it is imperative software is immune to crashes and data loss.

Lastly, developers have to satisfy **performance** quality requirement. Not only does the software have to be responsive, it has to maintain desirable performance across many different hardware configurations ranging from relatively modestly equipped university computers to high end workstations in an enterprise environment.

17.

Information Systems

Information systems deal with large amounts of information that can be in form of video, audio or traditional database.

Major qualities are

Integrity: Data accuracy must be preserved during storage, retrieval and manipulation.
Security: Data must be protected from unauthorized access both physically and virtually
Availability: Data must be stored in a redundant manner to be resistant to system crashes.

Critical systems

Critical system is any system that has potential to harm human life, economy, environment or some aspect of safety.

Major quality

Safety- *(explained above)

Distributed systems

Distributed systems are installed and executed on variety of operating systems and hardware configuration. These systems have a high degree or multiprocessing/parallel execution.

Major quality

Portability- *(explained above)

Real time systems

Real-time systems are types of systems that operate under time sensitive constraints. These systems respond to real-time events like mouse clicks or ignition timing in case of automotive engine control unit (ECU).

Major quality

Time efficiency- Data must be processes in specific time interval otherwise it is discarded regardless of its correctness.

Embedded systems

Embedded systems are sub-components of larger system and are generally specialized for their respective service. For example, inside of a car there are as many as 30 specialized computers that control various systems from entertainment to ABS, airbags to sunroof.

Major quality

Robustness- Many times embedded systems are safety critical and their firmware is not easily upgradable, for that reason, underlying software must be robust and correct.