**Scalability**

When it is stated that the Buzz system needs to be scalable, it refers to the ability to always have the appropriate amount of resources to handle the current workload on it. For Example: If the Buzz system were to run on a single instance of a server, that could for example handle 1000 concurrent connections, another instance of the server would have to be spawned as soon as the number of connections were to grow larger than 1000. If the amount of concurrent connections were then to decrease to a number less than 1000, the second instance of the server would have to stop, with all connections on it migrating to the first. In this way the system would always be running at maximum cost efficiency, as it would only use the amount of servers required at that specific time. This would be best implemented by taking a cloud computing approach to hosting the servers, as one can easily simply spawn additional instances of servers when required (without buying extra computers). The exact same approach should be taken for all storage done inside the system.

**Performance**

Since the buzz system is a web based application, the performance would be solely dependent on the response time to requests sent to it. Any response time longer than 1 second would interrupt the user’s thought flow, and would therefore not be acceptable. If a server is overloaded and the queue of requests is longer than the requests handled per second, a new server should be spawned to take over the queue from a certain point. If there is at any time cause for more than on server to be actively processing requests, there has to be load balancer that mitigates incoming requests between the active servers. When certain requests are sent to the System, that do not need to be handled immediately, such as getting usage statistics from server logs, these can be scheduled for handling at a time when server is under low strain. Since the persistent data in the system will be accessed on almost every request to the server, optimal structures and indexes are very important.

**Maintainability**

Since it is likely for the Buzz system to either become open source upon implementation, or be for sale for to external parties to integrate into their own systems, maintainability is of critical importance. For this reason an extremely modular approach has been taken in the design of the master specification. Due to the coupling ratio being kept as low as possible and the use of service contracts for all interactions between modules, said modules are pluggable. This makes the system extremely maintainable, since entire modules can be rewritten and plugged in if the need be. During development all programmers should follow strict coding standards, keep documentation up to date, and make complex algorithms as easy as possible to understand. During and after development all code should be tested rigorously to ensure reliability.