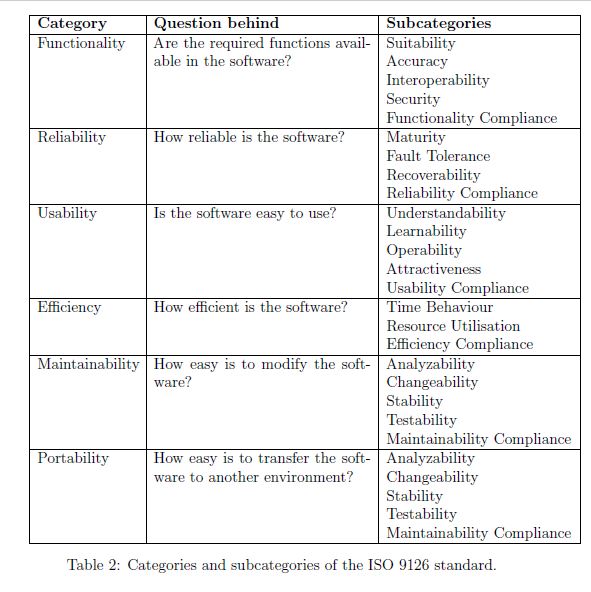
**Trade-offs in Software Quality Aspects of Twitter**

A trade-off is a situation in which you decrease one quality aspect in return for increasing another quality aspect. Ideally, software design decisions are made with full understanding of the positive and negative implications on its quality aspects. An example of a trade-off is a highly structured and modularized code that is easy to read by humans, hence easy to maintain. However, because it is highly structured it does not perform as well as less structured code.

In this case, a trade-off between Maintainability and Efficiency is made. An increase in Maintainability will decrease the Efficiency of a system and vice-verse. This situation can be illustrated by the sliders in figure 3.



****

Figure 3: The trade-off problem illustrated with sliders

The first slider shows a trade-off in favor of Efficiency, as the slider is set closer to Efficiency than to Maintainability. The second slider shows a trade-off in favor of Maintainability, as the slider is set close to Maintainability than to Efficiency. However, the second slider indicates that a lot of Efficiency is traded off against Maintainability. The first indicates a more neutral trade-off between Efficiency and Maintainability as the slider is positioned relatively close to the center of the bar.

**So the question here is “How reliable the twitter is?”,**

**Trade-off 1: Reliability Versus Maintainability**

To improve the reliability, the complexity of Twitter's server-infrastructure increased. They started to use more servers and they all had to be balanced and synchronized. Balancing and synchronizing servers using MySQL databases is not the easiest task, thereby reliability for maintainability.

They implemented a new feature called middle layer which is basically used as a queueing system to not overload the back-end service layer, as was the case before when they used no middle layer and only MySQL for storage. The middle layer was first implemented by Starling which is programmed in Ruby on Rails. But as this turned out to be too slow and with an unsatisfactory crash recovery. Then they used JRuby, RabbitMQ respectively with same false outcome. Finally, they used Scala. But they had to trade-off between maintainability and reliability since another language added to their long list of technologies they use for twitter.

Until 2012, running a service inside Twitter required hardware requisitions. Service owners had to find out and request the particular model or class of server, worry about your rack diversity, maintain scripts to deploy code, and manage dead hardware. So they migrated from bare metal environment to Mesos/Aurora which provides automatic rescheduling the instance on another host. This increased both maintainability, efficiency and reliability in twitter since generally each service that got their own bare metal hardware didn’t fully utilize its resources and did a poor job of managing capacity. Mesos allowed them to pack multiple services into a box without having to think about it, and adding capacity to a service is only requesting quota, changing one line of a config file, and doing a deploy.

**Trade-off 2: Reliability Versus Performance and Cost**

As twitter scaled their major workloads (Mesos, Hadoop, Manhattan, and MySQL) it became apparent the available market offerings didn’t quite meet the needs. Off-the-shelf servers come with enterprise features, like raid controllers and hot swap power supplies. These components improve reliability at small scale, but often decrease performance and increase cost; for example, some raid controllers interfered with the performance of SSDs and could be a third of the cost of the system.

**Trade-off 3: Reliability Versus Efficiency**

Twitter’s workload is divided into four main verticals: storage, compute, database, and gpu. Twitter defines requirements on a per vertical basis, allowing Hardware Engineering to produce a focused feature set for each. This approach allows them to optimize component selection where the equipment may go unused or underutilized. For example, their storage configuration has been designed specifically for Hadoop workloads and was delivered at a TCO reduction of 20% over the original OEM solution. At the same time, the design improved both the performance and reliability of the hardware. Similarly, for their compute vertical, the Hardware Engineering Team has improved the efficiency of these systems by removing unnecessary features.