Technical Solutions

Feasibility reports rely on analysing and comparing diverse technical solutions, as well as their implications and features. There are many factors to why technical solutions may be chosen, ranging from computational complexity to the potential training requirements for the solution to be implemented and/or utilised (Spacey, 2017).

This document is provided to give some criteria for analysis regarding technical solutions, in order to help decide between multiple options and identifying limitations with the options provided.

FUNCTIONALITY

Any technical solution will rely on functionality, a core component of functionality in software and computer systems. The functionality of a solution can be hard to measure; however, it describes the usage and limitations of a system, an essential part of designing and implementing a solution.

Functionality closely links with the requirements of a solution, outlined in a feasibility report. These requirements describe required functionality in the solution, explaining what features are necessary for completion, as well as describing potentially optional features, for example; features that include quality-of-life changes rather than the core functionality of the solution itself. (Bass, et al., 2012)

Analysing the functionality of different solutions allow for identification of potential issues with lacking features or undue complexity, as functionality that goes beyond what's required may also be detrimental to the core outcome due to the increased complexity and development time. (Paget, 2017)

COMPUTATIONAL COMPLEXITY

Leading on from functionality is computational complexity. Any technical solution with regards to programming and systems will have limitations and requirements in terms of system hardware and efficiency, relating to RAM usage, computational efficiency (including Landau notations) and hardware requirements (Ellinor, et al., n.d.).

Landau notation is important to consider as a part of computational efficiency as it helps describe scalability and complexity in a way that helps shape how technical solutions are produced and compared. Big-O notation describes how an algorithm, or mathematical function, scales in complexity based on the size of the input. O(n) describes a linear scaling, whereas $O(n^2)$ describes quadratic scaling (MIT, n.d.).

This difference in scaling means a linear solution becomes less difficult to execute than a quadratic solution as the input size rises, which is an important distinction when dealing with computational complexity and scalability.

Big-O notation is only a part in Landau notation, which describes not only the basic method of identifying scaling complexity, but also includes similar notations that are useful for *comparing* two functions, even if they're both linearly scaling.

Notation	Analogy
$f(n) = O(g(n))$	$\leq$
$f(n) = o(g(n))$	$<$
$f(n) = \Omega(g(n))$	$\geq$
$f(n) = \omega(g(n))$	$>$
$f(n) = \theta(g(n))$	$=$

(MIT, n.d.)

These notations describe comparisons between functions, such that $f(n) = o(g(n))$ means that function g scales slower than function f, meaning that $g(n)$ is less complex than $f(n)$, while still being a linear change. Similarly, $f(n) = \omega(g(n))$ means the inverse, such that $g(n)$ complexity grows faster than $f(n)$. This is very useful in comparing two different algorithms as it allows for a direct comparison of how the complexity scales with the input size for each function.

IMPLEMENTATION

The implementation of a technical solution should specify the systems proposed in a general scope, such as a server for hosting or storage, as well as the apps and software. As well as this, the implementation should describe any libraries or requirements utilised, including software modules like classes. This links with the computational complexity section, however it is broader in that it focusses on abstract objects, instead of on the specifics of the algorithms involved.

This abstract view of the modules and features allows for comparison in system complexity, allowing the comparison of overall systems and the requirements for server space, app development time, module development and support requirements. Should a solution require a login server, as well as a content server and a few load balancing servers, it would be more costly to implement than a peer-to-peer application with no requirements for servers (Waters, n.d.).

This means that the P2P option is likely going to be much cheaper and less complex to produce than the option that requires the server space. This direct comparison may not be useful in some cases, like those where solutions are very similar, however for diverse solutions it helps analyse the difference in a broad way.

RELIABILITY

Closely linked with the implementation considerations is the concern for reliability. Having a more complex server system means more points of failure, concerning server downtime and load requirements. Accounting for the differences in reliability in the system integration is a key component to comparing between different technical solutions, as it helps minimise risk of failure and downtime. Having the risks written down also allows for the feasibility report to later refer to said risks in order to help come up with contingencies.

Included in the reliability is the consideration of support requirements, concerning the requirement for updates, technical support, and long-term development and expansion of features. Having a less complex system with more reliable components lowers the risk of required bug fixes and support. Additionally, planning for future expansion of features helps determine whether any extra functionality should be added and considered in the future, in terms of risk and complexity.

REFERENCES

Bass, L., Clements, P. & Kazman, R., 2012. Chapter 4.2: Understanding Quality Attributes in Software Architecture. In: *Software Architecture in Practice, 3rd Edition*. s.l.:Addison-Wesley Professional.

Ellinor, A. et al., n.d.. *Big O Notation*. [Online]
Available at: https://brilliant.org/wiki/big-o-notation/
[Accessed 22nd November 2019].

MIT, n.d.. *Big O notation*. [Online]
Available at: https://web.mit.edu/16.070/www/lecture/big_o.pdf
[Accessed 22nd November 2019].

Paget, V., 2017. *Features vs Functionality: How to accurately compare software systems*. [Online]
Available at: https://blog.boardingware.com/features-vs-functionality/
[Accessed 22nd November 2019].

Spacey, J., 2017. *14 Types of Technical Feasibility*. [Online]
Available at: https://simplicable.com/new/technical-feasibility
[Accessed 22nd November 2019].

Waters, K., n.d.. *Peer-to-Peer vs. Client-Server Networks*. [Online]
Available at: https://www.techwalla.com/articles/peer-to-peer-vs-client-server-networks
[Accessed 22nd November 2019].