e-portfolio: https://sam-tselapedi.github.io/CyberSecurity/SSD.html

Secure Software Development: Reflective Essay

Information and cyber security is an essential component of any application that involves critical functionality. As digitalisation is increasingly becoming standard for modern businesses of various sizes and types, secure software development and security standards are increasingly essential. A secure software development life cycle (SSDLC) is a framework that prescribes how security should be built into application or software development from the outset and throughout. The privacy of individuals and the integrity of organisations whose responsibility it is to safeguard sensitive information is enormously threatened by dangers posed by data breaches.

Software quality and security are crucial attributes that need to be an endeavour at the initial stage of the development cycle. A software measured to be quality does not necessarily mean it is secure, while a software estimated to be secure does not necessarily imply it is of good quality. The two attributes are similar in their effects, and safe software is of a higher quality and security degree. That means the software functionalities are met: it prevents malicious actors and unauthorised processes from modifying it, and if it fails, it should do so safely in a predictable manner. Secure development lifecycle(SDL), a set of development practices for strengthening security and compliance, provides a structured approach to software and application security. These practices should be integral and integrated into all stages of software development and maintenance. Legislating and enforcing data protection is how governments worldwide enforce organisations to incorporate data protection safeguards at the earliest stages of development. For example, the European Unions' General Data Protection Regulation(GPDR) requires that data protection controls be integrated earlier in the settings of the software product and failing to comply or ignoring the regulation may result in hefty fines.

Verification and validation of software reduce defects attributable issues that lead to too poor quality software. Poor software quality can have adverse consequences such as critical infrastructure losses, even life. Consider the case of the Boeing 737 Max automation failure. Boeing designed an automated software tool called the Maneuvering Characteristics Augmentation System (MCAS) to automatically adjust the horizontal tail trim to stabilise the plane's pitch (Palmer, 2020). The system's failure resulted in the loss of life, and Boeing suffered a financial loss due to their aircraft being banned from flying in several countries. Such a case highlighted the significance of quality software, and yet too often, the time to the market and developments cost are afforded priority. Traceability is another essential attribute of software quality. Traceability implies that every requirement can be traced via documentation from the specification up to the specific line of code. And every line code can be traced up to the specification. It is essential to quality-approve every line of code at the earliest after it has been written to eliminate poor-quality code fragments in the final release. Poor quality code fragments may lead to vulnerabilities that may be lead to data leaks and potential exploitation by cyber intruders. Employing quality assurance practices often and regularly throughout the SDLC will assist with eliminating poor quality.

I have little to no formal experience in software development. For the more significant part of my career, I have had a substantial focus on network security with little knowledge of software security. Relating the knowledge I obtained from this module with my daily work activities, I now realise the significance of updating software security. Even if security was prioritised and integrated during the development, periodic updates are essential to maintaining and improving software's quality. Regularly testing software for flaws and vulnerabilities and repairing them before malicious actors discover them can protect an organisation from data breaches.

I observed a motivation cost factor in free-riding from one member of our group. Freeriding tends to erode the long-term motivation of other group members. In our group, it eventually led to a subtle conflict with the group. The solution I have learnt to address this in the future is to hold members accountable to the team contract because it was in agreement that we established clear expectations of the group members. Another challenge I faced with my group members was the poor attendance of seminars, which led to members not understanding what was required.

A critical example was when we needed to present our application in the last seminar. Non-attendance of the previous seminar where it was clearly stated what is required for the final seminar led to the presentation of our application been completed three days later. Being assessed as the group becomes a challenge when no mechanisms are provided to hold group members accountable to the team contract's agreed terms. I appreciated working as a tester while others functioned as developers because we discovered significant flaws and deviations from standards in our software project. Working in configuration enabled us to improve our coding and the supporting documents we have created.

My contribution to group work can be found on my e-portfolio.

References :

Hizazi, H. *et al.* (2014) 'Risk Factors in Software Development Projects', *Proceedings of the 6th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems*, 10(3), pp. 51–56.

Koc, G. and Aydos, M. (2017) 'Trustworthy scrum: Development of secure software with scrum', (c), pp. 244–249. doi: 10.1109/ubmk.2017.8093383.

Palmer, C. (2020) 'The Boeing 737 Max Saga: Automating Failure', *Engineering*, 6(1), pp. 2–3. doi: 10.1016/j.eng.2019.11.002.