**Adoption of a secure coding standard, and not leaving security to the end**

Secure coding standards are essential in producing secure and safe software which protects both the end user and the product owner. There are many foundations to adopting secure coding standards and implementing them. The foreground to that is using DevSecOps which is a development practice of integrating security into all phases of the SDLC. This ensures that security is built into every step and is not left to the end. This is beneficial for many reasons one way it keeps track of every item of software and what security is built alongside of it. It also allows for reduced long term costs as developers don’t have to play catch up to integrate software security. This also allows developers to implement automation and proper testing which will speed up the process while also being efficient and ensuring the testing was done correctly which then reduces chances for security vulnerabilities int eh software. Another key item to integrate is defense in depth which helps to add multiple layers of security to different aspects of the software. Although these are only a few examples integrating these will ensure security is built alongside software from the beginning and ensure the product is kept safe during the whole SDLC.

**Evaluation and assessment of risk and cost benefit of mitigation**

When running a company or creating products for sale there will always be a need for balance between cost and risk. As risk reduction practices go up the cost will also go up, but on the other hand as risk reduction implementation goes down so does cost and protection for software. Some of the added costs include additional workers and time needed to develop these items as well as possible hardware upgrades to keep up with the additional security measures such as stronger automation testing standards. On the other hand, companies can save money by not taking these steps but risk things being missed and security vulnerabilities. This is why it’s important to evaluate and success risk and then weigh it over how much security is needed. This should be part of the first step of the SDLC and again is included in proper usage of DevSecOps and shows the importance of implementing it. Once proper understanding is set it makes it easier to know how much security is necessary and then build on that foundation to overall prevent security vulnerabilities and threats.

**Zero trust**

Zero trust is the concept of assigning all users with no access initially and then granting them access to the part of the program they only need. This is important for many reasons which include easier accountability, easier initial setup of account accesses, and then security access audits can ensure the system is kept updated and tracked. Some other key principles to zero trust are proper usage of the “triple A” Authentication, authorization, and accountability this play a key role in zero trust and ensure trust is only given out when needed, taken away at appropriate times, and audits occur on a regular basis. Lastly, good zero trust practice is to always assume a breach and investigate every threat as a real threat. This ensures that real threats are dealt with, and all suspicions are looked into to prevent real attacks.

**Implementation and recommendations of security policies**

Security policies provide a rule book in theory to developers on how to program with security in mind, how to look out for issues, how to implement DevSecOps, Triple A, and Defense in depth. A security policy also helps to answer questions as to why these steps need to be implemented and can show the consequences to product owners who don’t see the value in implementing these steps. This overall helps to highlight potential threats, show ways to prevent these threats, and show the pros and cons for implementing them now vs later and show be implemented at the beginning stages of the SLDC.