Security requirements are a class of Non-Functional Requirements (NFRs) that relate to system confidentiality, integrity and availability. Explicitly stating security requirements during project inception is the perfect complement to security testing. Clearly outlining potential security requirements at the project onset allows development teams to make trade-offs about the cost of baking security into a project.

Like other NFR domains, there are two distinct classes of software security requirements:

**One Time:** A requirement that developers can implement, verify, and be sure that the requirement is complete. For example, hash and salt user passwords to protect against rainbow table attacks.

**Continuous:** A constraint on other requirements. These could affect a change to the system anytime. You need to integrate these constraints in the acceptance criteria of other requirements. For example, validate all HTTP post parameter values against valid character whitelists.

Consider the differences between these two types prior to integrating security requirements into your development process. In agile development, a user story based on a continuous requirement may be difficult to track. For example, “As a security analyst, I want to validate input on all HTTP post parameters” isn’t a user story that a single developer can realistically accomplish. It’s best suited as acceptance criteria in any user story that may result in new HTTP post parameters.

Like other requirements, the best security requirements are SMART:

**Specific:** Wording is clear and precise, not vague and all-encompassing. For example, “Protect against authentication threats” is not specific.

**Measureable / Testable:** There is a clear way to verify whether or not the requirement has been met. Note some NFRs need code review or static analysis because they may not be verifiable at runtime. This is contrast to most functional requirements.

**Actionable:** Developers reading the requirement have a clear understanding of what they need to do. For example, “Use defense in depth” is a useful principle, but not actionable.

**Realistic:** They take into account the real time, budget, and other resource constraints of the developers. For example, “Implement three-factor authentication” might result in a very secure system, but not be realistic for a development team to implement.

**Timely:** Has an upper-bounds on when the requirement needs to be implemented. This is arguably incompatible with agile development methods, so a more appropriate alternative is “Prioritized”. Prioritizing NFRs allows developers to judge how important a particular NFR is when weighed against functional requirements.